1 LE SYSTEME MSX

1.1 INTRODUCTION

Le système MSX n'a pas connu le succès escompté en Europe. Si l'on pouvait faire des reproches justifiés à la première version (prix trop élevé des premiers modèles, mémoire vive un peu juste, etc), il en va tout autrement de la version 2.

En effet, le MSX2 représente, en toute simplicité, le plus puissant ordinateur familial jamais construit ! Bien des machines professionnelles ne possèdent pas les caractéristiques du MSX2. Citons-en un, pour mémoire, quelques unes (prises sur le modèle le plus vendu) :

- 256 Ko de mémoire vive extensible à 4 Mo
- 128 Ko de mémoire écran extensible à 192 Ko
- 64 Ko de mémoire morte comprenant notamment un Basic Microsoft très évolué
- 2 slots d'extension, possibilité de passer à 8 slots
- lecteur de disquette 1 Mo (720 Ko formatée) intégré
- gestion de huit lecteurs de disquette simultanément
- compatibilité totale IBM PC au niveau des fichiers
- horloge interne alimentée par pile
- mémoire CMOS comprenant un système de mots de passe
- processeur graphique VLSI
- résolution 256 sur 212 pixels en 256 couleurs simultanément
- souris
- deux systèmes d'exploitation de disquettes (sont un possédant l'interface menus déroulants/icônes)
- générateur sonore sur trois voix, huit octaves
- 700 logiciels disponibles, dont une trentaine spécifique MSX2
- très nombreux périphériques

Il va de soi que la maîtrise du MSX2 ne se limite pas à une bonne connaissance du Z80 (micro-processeur qui équipe tous les MSX). Il est nécessaire, comme sur tous les ordinateurs, de pouvoir mettre en œuvre tous les composants du système. De plus, le programmeur qui travaille sur MSX doit veiller à maintenir la compatibilité, y compris avec les modèles futurs (MSX3) qui n'existent pas aujourd'hui. C'est le but de ce livre qui se propose de vous guider à travers l'exploration de la programmation du système MSX.

1.2 UTILISER CE LIVRE

Le livre que vous tenez entre vos mains a été conçu dans une double optique. Il aspire, dans un premier temps, à apprendre à tout programmeur ayant une bonne connaissance du Basic et des

notions sérieuses en Z80, à tirer le maximum de son ordinateur, qu'il s'agisse d'un MSX1 ou d'un MSX2. A ce propos, il est précisé à chaque fois que cela est nécessaire si les explications s'adressent uniquement au MSX2 ou aux deux ordinateurs. Puis, une fois les notions assimilées, le présent ouvrage a la prétention de servir de manuel de référence complétant ainsi le manuel d'utilisation fourni avec votre machine.

Comprendre le fonctionnement :

Pour chaque fonction, instruction, astuce, j'ai tenté de donner les explications les plus simples possibles. J'ai évité d'employer l'équivalent français de termes anglais lorsque ces derniers étaient usuels (je parle de « bitmap », ou de « flag » plutôt que d'indicateur binaire, mais j'emploie le mot « octet » et non « byte »). Lorsqu'une traduction paraissait hasardeuse, j'ai toujours donné le terme américain entre parnthèses.

De plus, suivant le vieil adage des informaticiens « rien ne vaut la pratique », presque toutes les explications se trouvent accompagnées d'un exemple de programme de Basic ou de langage machine l'illustrant. Lisez l'explication, si la compréhension n'est pas immédiate, essayez de taper le programme qui l'accompagne. Si vous ne voyez toujours pas, votre cas est sans espoir, cantonnez vous au Télé 7 jours.

Certains points ne sont pas abordés dans ce livre. Il s'agit tout d'abord du micro-processeur. Si vous avez des problèmes avec cet élément, je ne peux que vous conseiller l'achat de « Programmation du Z80 »de Rodnay Zaks chez Sybex, véritable « Bible » du Z80 de plus de six cent pages. De même, je ne m'étends pas sur le fonctionnement du processeur sonore PSG (« Programmable Sound Generator »). Tous les manuels livrés avec les différents modèles de MSX donnent les renseignements nécessaires à la programmation de ce composant. Voyez tout de même le paragraphe 6.6, « Faire de la musique en langage machine », du chapitre 6.

Ce manuel ne comporte aucune information sur le matériel même (« hardware »). Il n'aborde les différents éléments qu'au niveau logiciel (« software »). Il s'adresse donc avant tout au programmeur et non à l'électronicien.

Retrouver un renseignement rapidement :

Lorsque vous maîtrisez l'application qui vous intéresse, évitez tout ce qui est note, remarque, etc. En général, les renseignements indispensables à la mise en œuvre de l'application (adresses mémoires, registres à charger, etc) se trouvent en début de paragraphe, les explications venant après. Il va de soi qu'il peut être utile de consulter les programmes donnés en exemple, ils permettront en effet souvent un gain de temps appréciable.

A la fin de l'ouvrage sont réunies six annexes qui renferment une grande partie des petites choses dont un programmeur a toujours besoin et qui ne se trouvent, bien entendu, jamais là où on les cherche. Essayez d'exploiter ces annexes au maximum.

Dans tous les cas, n'hésitez pas à corriger les erreurs et à compléter les oublis qui ne manqueront pas de se révéler. Si vous avez le temps, envoyez moi tous les conseils, remarques et autres critiques que la lecture de cet ouvrage vous aura inspirés. Je vous en remercie d'avance.

1.3 CONSEILS AU DEVELOPPEUR DE LOGICIELS

- Les programmes que vous écrivez ne doivent en aucun cas adresser directement la mémoire morte. Celle-ci varie d'une marque à l'autre et surtout des MSX1 aux MSX2 en général. Il est absolument indispensable de passer par la table des sauts de 00000H à 001F9H pour utiliser le Bios.
- Pour maintenir la compatibilité entre les différents MSX, il ne faut en aucun cas accéder au matériel sans passer par le Bios. Ce dernier constitue une sorte de tampon entre les programmes et le matériel.

  La seule exception à la règle ci-dessus concerne le processeur vidéo. Pour des raisons de vitesse d'exécution, certains programmes peuvent adresser le VDP (« Video Display Processor ») sans passer par le Bios. Voir le chapitre 5 pour plus de précisions à ce sujet.

  Les adresses mémoire 00004H et 00005H contiennent l'adresse du générateur de caractères (CGTABL).
- Ne pas utiliser la mémoire vive située au dessus de 0F380H comme de la RAM ordinaire. En effet, cette zone mémoire contient les variables système (voir chapitre 4) indispensables à la bonne marche de votre ordinateur.
- Il existe des différences entre les MSX commercialisés en France et dans les autres pays - ne serait-ce que le clavier - dont il est parfois important de tenir compte. Les cases mémoire 0002BH et 0002CH donnent des renseignements importants à ce sujet (voir le chapitre 3).

  Sur MSX2 uniquement, vous trouverez des renseignements complémentaires dans la mémoire CMOS de l'horloge. Voir la routine REDCLK (001F5H), chapitre 3, paragraphe 3.3 « Le Bios en Sub-ROM ».
- Certains programmeurs placent la pile en haut de mémoire avec l'instruction LD SP, 00000H. Ceci ne fonctionne évidemment pas sur MSX. L'adresse 0FFFFH contient justement des renseignements précieux sur l'état des slots (voir le chapitre 2 à propos des slots).
- Certains programmeurs partent du principe que la mémoire vive principale ou vidéo contient 0 à l'allumage. Il va de soi que ce n'est pas le cas et que la RAM peut contenir à peu près n'importe quoi si elle n'a pas été modifiée lors de l'initialisation du système.
- Si votre programme ne peut fonctionner avec la présence d'un lecteur de disquettes, il suffit de vérifier la présence du lecteur (voir le chapitre 6) puis, si un lecteur est bien présent, d'afficher à l'écran le message « Faites un RESET en laissant la touche SHIFT enfoncée jusqu'au bip sonore ».

  En effet, lors de l'initialisation du système, si le MSX détecte que la touche SHIFT est enfoncée, il ne valide pas le lecteur de disquettes.

  Sur le même principe, la touche CTRL assigne un seul lecteur par contrôleur. Si vous avez 2 lecteurs de disquettes avec 2 contrôleurs, lors de toutes les opérations, le premier s'appelle « A » alors que le second prend la dénomination « C ». En enfonçant CTRL à l'initialisation, vous gagnerez de la place mémoire et vos drives se nommeront « A » et « B ».
- Lorsque l'on écrit dans un registre « Write only » du processeur vidéo, il est préférable de sauvegarder la donnée dans la zone des variables systèmes adéquate. Ainsi, la donnée pourra être relue à tout moment, ce qui serait impossible autrement.

1.4 ET MAINTENANT...

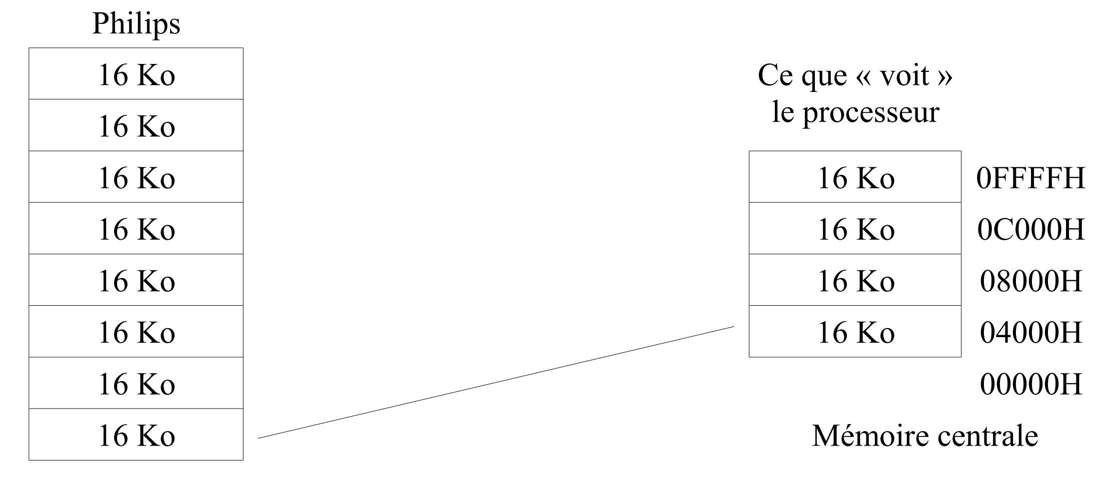
A présent, vous allez vous lancer dans la découverte des slots, Bios, variables systèmes, et autres processeurs vidéo. Une bonne dose de patience sera, dans la plupart des cas, appréciable et bénéfique. Et maintenant à vous de jouer... et bonne chance !

2 LES SLOTS ET LE MEMORY MAPPER

2.1 COMMENT DEPASSER LA LIMITE DES 64 KO

Tous les ordinateurs MSX sont équipés d'un Z80 comme micro-processeur principal. Ce dernier est un processeur « 8 bits », ceci signifie qu'il manipule les données par paquets de 8. Quant aux adresses mémoires, elles se trouvent codées sur 2 mots de 8 bits (deux octets), soit 16 bits. Les adresses peuvent donc prendre n'importe quelle valeur entre 0000000000000000 et 1111111111111111 en binaire, ou 0 et FFFF en hexadécimal. Ceci équivaut en décimal à une valeur entre 0 et 65535. Sachant qu'un « kilo » informatique ne vaut pas 1000 mais 2 puissance 10 - ou 1024 - octets (un des petits secrets qui fait le charme de l'informatique), le Z80 qui peut accéder à 65536 adresses, gère donc 65536/1024, soit 64 kilo octets.

Ainsi, quoiqu'il arrive, le Z80 ne pourra jamais « voir » plus de l4 Ko de mémoire. Ceci étant, il est rare qu'une application nécessite plus de 64 Ko d'un seul tenant. Les concepteurs du MSX ont donc introduit un système qui permet de définir sur quel bloc de 64 Ko le Z80 va travailler. En réalité, l'utilisateur peut choisir un bloc de 16 Ko.



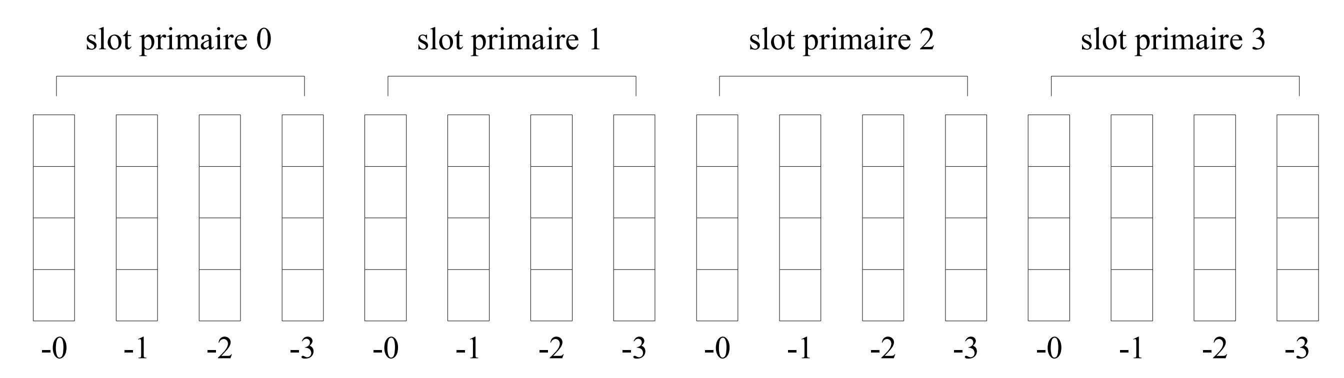
Toutes les opérations de manipulations de blocs restent invisibles au Z80 qui utilise ses 64 Ko comme s'il n'y avait pas d'autre mémoire.

Un bloc de 16 Ko s'appelle une « page ». La mémoire centrale est donc constituée de 4 pages (pages 0 à 3). Il existe deux dispositifs autorisant la manipulation de pages : les slots (sur MSX1 et MSX2) et le « Memory Mapper » (certains MSX2 uniquement).

2.2 QU'EST-CE QU'UN SLOT ?

Le système de slot est de loin le plus répandu pour gérer la mémoire sur les MSX. En fait un slot (ou fente) primaire correspond à un port cartouche, il s'agit donc d'un dispositif matériel (« hardware »). A l'opposé, les slots secondaires reposent sur un montage logiciel (« software »), qui permet d'étendre la mémoire dans les slots primaires qui ne recevront jamais de cartouche (le slot primaire 0, par exemple, est interne et, ne possédant pas de connecteur, il ne pourra jamais accueillir une cartouche). Chaque slot primaire contient 4 slots secondaires, chacun de ceux-ci renfermant 4

pages de 16 Ko.



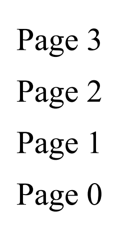
Au niveau du programmeur, nul besoin de faire la différence entre slot primaire et secondaire, il suffit de considérer qu'il y a 16 slots (appelés « slots secondaires »), les quatre premiers étant numérotés 0-0, 0-1, 0-2 et 0-3 ! Les quatre suivants seront donc les slots secondaires 1-0, 1-1, 1-2, 1-3 et ainsi de suite.

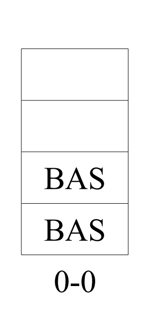
La mémoire centrale sera déterminée par le numéro de slot pour chaque page :

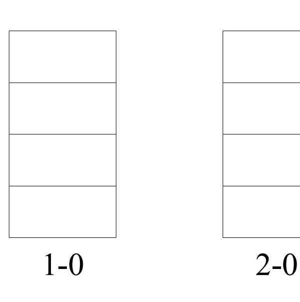
3	3-2	on prend la page 3 du slot 3-2 comme page 3 du Z80
2	1-3	on prend la page 2 du slot 1-3 comme page 2 du Z80
1	3-0	on prend la page 1 du slot 3-0 comme page 1 du Z80
0	0-0	on prend la page 0 du slot 0-0 comme page 0 du Z80

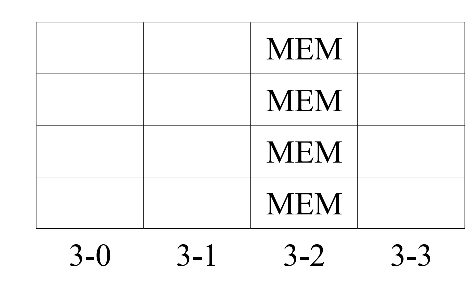
Mémoire centrale

Voici un exemple de disposition sur un MSX1, le VG 8020/19 de Philips :









Quelques remarques :

- Les pages 0 et 1 du slot 0-0 contiennent l'interpréteur Basic et le Bios en mémoire morte.
- Les slots 1-0 et 2-0 correspondent aux deux ports cartouches du 8020. Un jeu en cartouche pourra être lu (et copié) en lisant le contenu du slot 1-0 ou 2-0.
- On comprend pourquoi on ne dispose que de 32 Ko (même un peu moins) sous Basic. Le Z80 « voit » l'interpréteur Basic en pages 0 et 1. Il ne reste donc que les pages 2 et 3 pour de la mémoire vive.
- Lorsqu'il n'y a qu'un slot secondaire dans un slot primaire, celui-ci peut prendre n'importe quel numéro. Par exemple, ici, on peut accéder au slot 0-0 en l'appelant 0-0 aussi bien que 0-1, 0-2 ou 0-3. On peut même le désigner tout simplement comme le slot 0. Il en va de même pour les

slots 1-0 et 2-0 dans l'exemple du 8020. Par contre, on doit obligatoirement utiliser le numéro 3-2 pour la mémoire vive, même s'il n'y a rien dans les autres slots. C'est pour cette raison qu'il est préférable de toujours prendre la dénomination en slot secondaire (slot 0-0) et non primaire (slot 0).

Voici un exemple de disposition sur un MSX2, le VG 8235 de Philips :

	0-0	1-0	2-0	3-0	3-1	3-2	3-3
Page 3						MEM	
Page 2						MEM	
Page 1	BAS					MEM	DOS
Page 0	BAS			SUB		MEM	

Quelques remarques :

- Les pages 0 et 1 du slot 0-0 contiennent le MSX-Basic et le Bios en mémoire morte. La page 0 du slot 3-0 renferme la Sub-ROM (ou ROM auxiliaire) qui contient la plupart des instructions supplémentaires du Basic version 2.0 par rapport au Basic 1.0 du MSX1. Les pages 0 à 3 du slot 3-2 contiennent 64 Ko de mémoire vive (les autres 64 Ko de la mémoire vive totale qui compte 128 Ko sur le 8235 sont présents dans le memory mapper que nous verrons plus loin dans ce chapitre). Les fonctions du Disk-Basic (le 8235 est équipé d'un lecteur de disquettes intégré) se trouvent en mémoire morte en page 1 du slot 3-3.
- Les slots 1-0 et 2-0 correspondent aux deux ports cartouches.

La configuration minimum que l'on est certain de trouver sur tout ordinateur se compose de :

sur MSX1 :

- une page de mémoire vive de 0C000H à 0FFFFH dans n'importe quel slot
- deux pages (32 Ko) de mémoire morte contenant le MSX-Basic version 1.0 et le Bios.
- un port cartouche occupant un slot primaire

sur MSX2 :

- quatre pages consécutives (64 Ko) de mémoire vive dans n'importe quel slot.
- trois pages (48 Ko) de mémoire morte divisée en deux parties (Main-ROM et Sub-ROM), dans n'importe quel slot.
- deux ports cartouches occupant deux slots primaires.

2.3 UTILISER LES SLOTS

Dans toutes les routines du Bios et variables systèmes, le numéro d'un slot est toujours codé de la même manière :

7	6	5	4	3	2	1	0
EXT	??2	??1	??0	SS1	SS0	PP1	PP0

SS1 et SS0 : ces deux bits donnent le numéro du slot secondaire (de 0 à 3).

PP1 et PP0 : alors que ceux-ci définissent le slot primaire (de 0 à 3)

??0 à ??2 : ces bits varient suivant le contexte, dans tous les cas, ils n'ont aucun rapport avec les slots.

EXT : le bit de poids fort de l'octet est un flag qui indique qu'il s'agit d'un slot secondaire (EXT à 1), ou alors d'un slot primaire (EXT à 0).

Les variables systèmes liées aux slots :

- EXPTBL (0FCC1H) type : MSX1/MSX2

  cette variable système contient quatre octets, chaque octet indique si le slot primaire correspondant est étendu en slot secondaire (l'octet est 080H) ou s'il ne l'est pas (octet à 0).

- SLTTBL (0FCC5H) type : MSX1/MSX2

  SLTTBL, constituée de quatre octets aussi, retient en parallèle de EXPTBL la valeur du registre d'extension lorsqu'un slot primaire est étendu (080H).

- SLTATR (0FCC9H) type : MSX1/MSX2

  les 64 octets de SLTATR correspondent chacun à une page de 16 Ko de mémoire (16 slots secondaires fois 4 pages donnent bien 64 pages). L'octet donne le type d'extension pour chaque page (0=pas d'extension).

  Les 5 bits de poids faible ne sont pas utilisés. Les 3 bits de poids fort suivent le code :

7	6	5	4	3	2	1	0
BT	DE	SE	?	?	?	?	?

BT : bit « Basic Text », programme Basic

DE : « Device Expander », extension matériel

SE : « Statement Expander », extension logiciel

Voici un petit programme Basic qui donne un descriptif de votre ordinateur :

```
10 SCREEN 0:WIDTH80:COLOR 10,0,0:CLS
20 FOR I=&HFCC9 TO &HFD08
30 IF PEEK(I) = 0 THEN NEXT:END
40 A=I-&HFCC9
50 PRINT « Page » + STR$(A MOD4) + SPACE$(2) + « slot » + STR$(INT(A/16)) + « - » + STR$
((A/4) MOD 4) + « : »;
60 IF (PEEK(I) AND &H80)=128 THEN PRINT TAB(20) + « Extension Basic. »
70 IF (PEEK(I) AND &H40)=64 THEN PRINT TAB(20) + « Extension Matériel. »
80 IF (PEEK(I) AND &H20)=32 THEN PRINT TAB(20) + « Extension Logiciel. »
90 PRINT:NEXT
```

– SLTWRK (0FD09H) type : MSX1/MSX2

Les 128 octets de SLTWRK constituent une zone de travail. Deux octets sont réservés à chacune des 64 pages.

– MNROM (0FCC1H) type : MSX2

Sur MSX2, cet octet contient le numéro de slot qui contient l'interpréteur Basic et le Bios principal.

– SUBROM (0FAF8H) type : MSX2

L'adresse 0FAF8H donne le numéro du slot qui contient la Sub-ROM.

Les routines du Bios liées aux slots :

– ReaD SLoT

Nom : RDSLT
Adresse : 0000CH / ROM
Rôle : sélection d'un slot et lecture d'une case mémoire.

– WRite SLoT

Nom : WRSLT
Adresse : 00014H / ROM
Rôle : sélection d'un slot, puis écriture dans une case mémoire.

– CALl SLoT

Nom : CALSLT

Adresse : 0001CH / ROM

Rôle : appel inter-slot à une adresse.

- ENAble SLoT

Nom : ENASLT

Adresse : 00024H / ROM

Rôle : sélection d'un slot.

- CALL Far

Nom : CALLF

Adresse : 00030H / ROM

Rôle : appel inter-slot à une adresse.

Ces routines n'appellent pas de commentaire particulier (voir le Bios) sauf peut-être la dernière routine « CALL Far » :

Pour accéder à une adresse se trouvant dans un autre slot, il suffit d'écrire les 3 instructions suivantes en assembleur :

RST 30

DB numéro du slot

DW adresse à appeler

Le RET de la routine appelée renverra l'exécution à l'octet immédiatement après l'adresse définie par le DW. Par exemple la suite d'octets F7, 8E, 0, C0, AF génèrerait la séquence suivante :

0F7H code Z80 de « RST 30 »

08EH slot secondaire 2-3

000H adresse à appeler 0C000H

0C0H

0AFH code Z80 de « XOR A », le RET de la routine en 0C0000H dans le slot 2-3 fera reprendre l'exécution du programme avec ce « XOR A ».

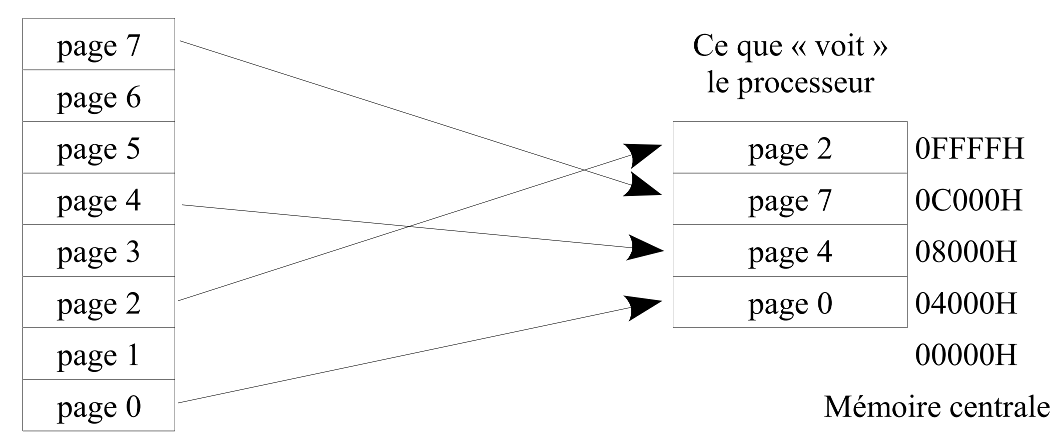
2.4 LE MEMORY MAPPER (MSX2)

Certains MSX2 disponibles en France se trouvent équipés d'un dispositif perfectionné permettant de gérer très facilement des quantités assez importantes de mémoire vive (128 ou 256 Ko de RAM sur les modèles disponibles chez nous, mais pouvant aller en théorie jusqu'à 4 Mo).

Vous avez sans doute remarqué que le principal défaut du système de slots réside dans l'impossibilité de changer de page. Ainsi la page 2, par exemple, peut être choisie dans n'importe quel slot, dison le slot 3-1. Cependant, il est impossible de prendre la page 0 su slot 3-1 comme page 2 en mémoire centrale.

Avec le memory mapper, ce genre de problème n'existe plus.

Voyons le schéma d'une mémoire vive totale de 128 Ko « memory mappée » :



On parle de page « logique » pour la mémoire contenue dans le memory mapper, et de page « physique » pour la mémoire centrale. Pour chacune des 4 pages physiques de la mémoire centrale, on donne le numéro d'une page logique du memory mapper. On peut donc mettre n'importe quelle page logique n'importe où. Il est également possible de dupliquer des pages, etc. Le memory mapper contient en général au minimum 7 pages logiques (128 Ko), mais peut aller jusqu'à 256 pages (4096 Ko).

En utilisation normale, le programmeur ne touchera pas à la page physique 0 (Bios), ni à la page physique 3 (variables systèmes). Par contre, il pourra manipuler les pages physiques 1 et 2 à loisir.

<u>Utilisation du memory mapper</u> :

La gestion du memory mapper est particulièrement simple : on utilise les quatre ports d'entrée/sortie, de 0FCH à 0FFH.

En effet, 0FCH contient la page logique du memory mapper correspondant à la page physique 0. Le port 0FDH donne la page logique pour la page physique 1, le port 0FEH code (l'aviez-vous deviné ?) la page logique utilisée pour la page physique 2. Le contenu de 0FFH précisant la page logique que l'on associe à la page physique 3.

0FCH	page logique pour page physique 0
0FDH	page logique pour page physique 1
0FEH	page logique pour page physique 2
0FFH	page logique pour page physique 3

Ces ports restent accessibles aussi bien en écriture qu'en lecture (sous Basic, le bit de poids fort est toujours à 1). Si votre MSX ne possède pas de memory mapper, ces ports peuvent contenir 0F8H (ou autre chose) mais ils seront sûrement tous identiques.

A l'initialisation, le MSX sélectionne les pages de la manière suivante :

mémoire centrale	port I/O	contenu du port	page logique
page 0	0FCH	003H	page 3
page 1	0FDH	002H	page 2
page 2	0FEH	001H	page 1
page 3	0FFH	000H	page 0

Comme le memory mapper est une option sur MSX2, il n'existe aucune variable système ou routine du Bios, le concernant. Si vous désirez utiliser le memory mapper, votre programme devra déterminer seul la présence ou absence du memory mapper ainsi que sa taille mémoire.

Microsoft demande à toute société commercialisant des logiciels MSX2 utilisant le memory mapper de porter sur l'emballage la mention « Requires XXX K MEMORY MAPPER ».

3 LE BIOS (BASIC INPUT/OUTPUT SYSTEM)

3.1 INTRODUCTION AU BIOS

Le Bios (Basic Input/Output System) a été conçu afin de permettre au programmeur d'accéder aux routines préprogrammées de la ROM sans que cette dernière soit toujours la même. En effet, le Bios constitue une sorte de tampon entre l'utilisateur et le matériel (« harware »). Actuellement, le clavier du MSX est géré par un circuit spécialisé : le PPI 8255. A l'avenir, il se peut qu'un constructeur MSX sorte un modèle avec clavier détaché à liaison infrarouge. Il coule de source que ce nouvel ordinateur ne sera pas muni d'un 8255? Si un programme fait des accès directs au PPI, il ne fonctionnera pas correctement sur le nouveau modèle. Au contraire, si le programme passe par la routine « lire l'état du clavier », il tournera très bien avec le clavier à liaison infrarouge. Le Bios concilie compatibilité des logiciels avec évolution du matériel.

3.2 LE BIOS EN MAIN-ROM

Vous trouverez pour chaque routine du Bios en mémoire centrale son nom et son adresse mémoire, son type (pour les possesseurs de MSX1), les paramètres devant être fournis à la routine, les résultats que vous récupèrerez ainsi que les registres modifiés et parfois une remarque sur la routine elle-même ou sur son fonctionnement.

A la rubrique « type », la désignation MSX1 précise que la routine existait déjà sur MSX1, elle reste bien entendu compatible avec le MSX2. Dans ce cas de figure, un chiffre précise le genre de modifications apporté sur MSX2 par rapport à l'ancienne routine. Voici la table des codes :

1 - Aucune modification par rapport à la routine MSX1.

2 - Appel à la Sub-ROM si l'écran est en mode 5, 6, 7 ou 8.

3 - Appel systématique à la Sub-ROM.

4 - Routinie modifiée pour pouvoir traiter les cas des modes d'écrans 4 à 8.

Note : Les RST 0 à 5 sont réservés au Basic. Le RST 6 est utilisé pour les appels inter-slots, le RST 7 pour les interruptions « hard ».

00000H **CHKRAM** (RST 0) CHecK RAM

Nom :	CHKRAM
Adresse :	00000H/ROM
Type :	MSX1 - 1 -
Rôle :	redémarrage complet du système. Recherche de la RAM, mise en place des slots, etc.
Entrée :	rien
Sortie :	rien
Modifie :	tout

00004H **CGTABL** Character Generator TABLe

Nom :	CGTABL
Adresse :	00004H/ROM
Type :	MSX1 Donnée
Rôle :	les octets 00004H et 00005H contiennent l'adresse du début du générateur de caractères en ROM

00006H **VDP.DR** VDP Data Read port

Nom :	VDP.DR
Adresse :	00006H/ROM
Type :	MSX1 Donnée
Rôle :	cet octet renferme l'adresse du port de lecture du processeur vidéo. Certains logiciels nécessitant des accès vidéo très rapides peuvent adresser directement le VDP (sans passer par le BIOS) à condition de venir lire cet octet avant tout IN

00007H **VDP.DW** VDP Data Write port

Nom :	VDP.DW
Adresse :	00007H/ROM
Rôle :	cet octet renferme l'adresse du port d'écriture du processeur vidéo. Voir explication ci-dessus.

00008H **SYNCHR** (RST 8) SYNtax of CHaRacter

Nom :	SYNCHR
Adresse :	00008/ROM
Type :	MSX1 - 1 -
Rôle :	vérification de l'octet suivant le RST 8. Si c'est le même que celui pointé par HL, saut en CHRGTR, sinon affichage de « Syntax error »
Entrée :	HL - caractère actuel
	Octet à vérifier après le RST
Sortie :	HL - caractère suivant
	A - caractère actuel
	indicateur carry à 1 - c'est un nombre
	indicateur zéro à 1 - fin d'instruction
Modifie :	AF, HL
Note :	Ce RST est réservé au Basic

0000CH **RDSLT** ReaD SLoT

Nom :	RDSLT
Adresse :	0000CH/ROM
Type :	MSX1 - 1 -
Rôle :	sélection d'un slot et lecture d'une case mémoire.
Entrée :	HL - adresse de la case mémoire à lire
	A - numéro du slot sous la forme binaire F000SSPP
	F : flag 0 si slot primaire
	flag 1 si slot secondaire
	SS : numéro de slot secondaire (0-3)
	PP numéro de slot primaire (0-3)
Sortie :	A - contenu de la case mémoire
Modifie :	AF, BC, DE
Note :	Interruptions automatiquement interdites lorsque l'on appelle cette routine.
	Existe aussi sous MSX-DOS (peut être appelé sous MSX-DOS sans changer de slot)

00010H **CHRGTR** (RST 10)

Nom :	CHRGTR
Adresse :	00010H/ROM
Type :	MSX1 - 1 -
Rôle :	Récupération du caractère suivant sous Basic
Entrée :	HL
Sortie :	HL pointe sur le caractère suivant
Modifie :	AF, HL
Note :	Ce RST est réservé au Basic

00014H **WRSLT** WRite SLoT

Nom :	WRSLT
Adresse :	00014H/ROM
Type :	MSX1 - 1 -
Rôle :	sélection d'un slot, puis écriture dans une case mémoire
Entrée :	HL - adresse de la case mémoire
	E - donnée à écrire
	A - numéro du slot sous la forme F000SSPP
	F : flag 0 si slot primaire
	1 si slot secondaire
	SS : numéro de slot secondaire (0-3)
	PP : numéro de slot primaire (0-3)
Sortie :	Rien
Modifie :	AF, BC, D
Note :	Interruptions automatiquement interdites.
	Cette routine existe aussi sous MSX-DOS.

00018H **OUTDO** (RST 18) OUT DO

Nom : OUTDO

Adresse : 00018H/ROM

Type : MSX1 - 1 -

Rôle : sortie d'un caractère alphanumérique sur écran, imprimante ou disquette .

Entrée : A - code ASCII du caractère à sortir

PRTFLG - flag, 1 si sortie sur imprimante

PTRFIL - si différent de 0, sortie sur disquette dans le fichier pointé par PTRFIL

Sortie : rien

Modifie : rien

Note : Appel au hook H.OUTD

Ce RST est réservé au Basic

0001CH **CALSLT** CALl SLoT

Nom : CALSLT

Adresse : 0001CH

Type : MSX1 - 1 -

Rôle : appel inter-slot à une adresse

Entrée : IX - adresse à appeler

IY - numéro du slot sous la forme F000SSPP

F : flag 0 si slot primaire

1 si slot secondaire

SS : numéro de slot secondaire (0-3)

PP : numéro de slot primaire (0-3)

Sortie : Rien

Modifie AF, IX, IY, registres auxiliaires

Note : Interruptions automatiquement interdites.

Cette routine existe aussi sous MSX-DOS.

Ne jamais passer d'arguments à un sous-programme via les registres auxiliaires du Z80 quand vous utilisez CALSTL.

00020H **DCOMPR** (RST 20) Double register COMpare

Nom : DCOMPR
Adresse : 00020H/ROM
Type : MSX1 - 1 -
Rôle : Comparaison de [HL] et [DE]
Entrée : HL - premier nombre
DE - deuxième nombre
Sortie : F - état des indicateurs
Modifie : AF
Note : indicateur carry à 1 - [HL]<[DE]
indicateur zéro à 1 - [HL]=[DE]

00024H **ENASLT** ENAble SLoT

Nom : ENASLT
Adresse : 00024H/ROM
Type : MSX1 - 1 -
Rôle : sélection d'un slot
Entrée : HL - page à sélectionner (0, 04000H, 08000H ou 0C000H)
A - numéro du slot sous la forme F000SSPP
F : flag 0 si slot primaire
1 si slot secondaire
SS : numéro de slot secondaire (0-3)
PP : numéro de slot primaire
Sortie : rien
Modifie : tout
Note : Interruptions automatiquement interdites.
Cette routine existe aussi sous MSX-DOS.
Exemple d'utilisation : passage de MSX-DOS au Basic

```
        ORG   0D000H
MNROM   EQU   0FCC1H
ENASLT  EQU   00024H
RETURN  EQU   0409BH
DEBUT:  LD A,(MNROM)
        LD HL, 0
        CALL ENASLT     ; Bios en page 0
        LD A, (MNROM)
```

```
LD HL,04000H
CALL ENASLT        ; Basic page 1
JP RETURN          ; saut à l'interpréteur
END DEBUT
```

00028H **GETYPR** (RST 28)

Nom : GETYPR

Adresse : 00028H/ROM

Type : MSX1 - 1 -

Rôle : Détermination du type de FAC

Entrée : FAC

Sortie : F - état des indicateurs

Modifie : AF

Note : Ce RST est réservé au Basic.

0002BH **BASRVN** BASic Rom Version Number

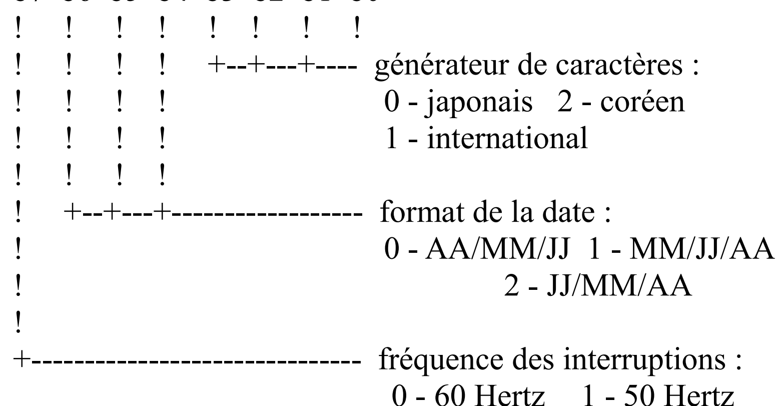
Nom : BASVRN

Adresse : 0002BH/ROM

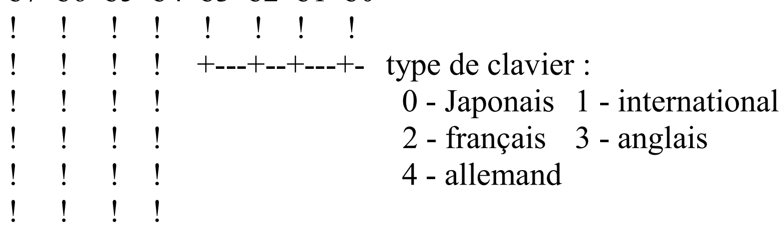
Type : MSX1 Donnée

Rôle : Ces deux octets contiennent tous les renseignements utiles à des programmes destinés à fonctionner dans des pays différents (« international software »).

0002BH



0002CH



0002DH **MSXVER** MSX VERsion

Nom :	MSXVER
Adresse :	0002DH/ROM
Type :	MSX2 donnée
Rôle :	version de MSX, contient 0 pour un MSX1, 1 pour un MSX2
Note :	Les autres numéros sont réservés pour les versions futures de MSX.

00030H **CALLF** (RST 30) CALL Far

Nom :	CALLF
Adresse :	00030H/ROM
Type :	MSX1 - 1 -
Rôle :	appel inter-slot à une adresse
Entrée :	rien
Sortie :	rien
Modifie :	AF, IX, IY, registres auxiliaires.
Note :	CALLF diffère de CALSTL dans la manière d'appeler la routine. Pour CALLF il faut écrire :

```
RST  30
DB   numéro du slot sous la forme F000SSPP
DW   adresse à appeler
```

Interruptions automatiquement interdites.

Cette routine existe aussi sous MSX-DOS.

Ne jamais passer d'arguments à un sous-programme via les registres auxiliaires du Z80 lorsque vous utilisez CALLF.

00038H **KEYINT** (RST 38) encode KEYboard / timer INTerrupt routine

Nom :	KEYINT
Adresse :	00038H/ROM
Type :	MSX1 - 1 -
Rôle :	Interruption hard.
Entrée :	Rien
Sortie :	Rien
Modifie :	Rien
Note :	Cette routine appelle les hooks H.KEY1 et H.TIMI

Appel à la Sub-ROM pour les conversions roma-kana

Ce RST est réservé aux interruptions hard

0003BH **INITIO** INITialize Input/Output

Nom : INITIO
Adresse : 0003BH/ROM
Type : MSX1 - 1 -
Rôle : initialisation du PSG pour les entrées/sorties
Entrée : rien
Sortie : rien
Modifie : tout

0003EH **INIFNK** INItialize FuNction Key

Nom : INIFNK
Adresse : 0003EH/ROM
Type : MSX1 - 1 -
Rôle : ré initialisation des touches de fonction à leur valeur de départ.
Entrée : rien
Sortie : rien
Modifie : tout

00041H **DISSCR** DISable SCReen display

Nom : DISSCR
Adresse : 00041H/ROM
Type : MSX1 - 1 -
Rôle : Extinction de l'écran.
Entrée : rien
Sortie : rien
Modifie : AF, BC

00044H **ENASCR** ENAble SCReen display

Nom : ENASCR
Adresse : 00044H/ROM
Type : MSX1 - 1 -
Rôle : Allumage de l'écran.
Entrée : rien
Sortie : rien
Modifie : AF, BC

00047H **WRTVDP** WRiTe VDP

Nom :	WRTVDP
Adresse :	00047H/ROM
Type :	MSX1 - 2 -
Rôle :	écriture dans un registre du VDP
Entrée :	C - numéro du registre du VDP (0-23 et 32-46)
	B - valeur à écrire
Sortie :	rien
Modifie :	AF, B
Note :	appel à la Sub-ROM si le bit EV du registre 0 du VDP est modifié ou si le numéro de registre est compris entre 8 et 46.

0004AH **RDVRM** ReaD VRaM

Nom :	RDVRM
Adresse :	0004AH/ROM
Type :	MSX1 - 1 -
Rôle :	lecture d'une case mémoire de la VRAM
Entrée :	HL - adresse de la case mémoire à lire
Sortie :	A - contenu de la case mémoire lue
Modifie :	AF
Note :	pour lire une case mémoire de VRAM dont l'adresse est supérieure à 03FFFH, utilisez la routine NRDVRM.

0004DH **WRTVRM** WRiTe VRaM

Nom :	WRTVRM
Adresse :	0004DH/ROM
Type :	MSX1 - 1 -
Rôle :	écriture dans une case mémoire de la VRAM
Entrée :	HL - adresse de la case mémoire (00000H - 03FFFFH)
	A - donnée à écrire
Sortie :	rien
Modifie :	AF
Note :	pour écrire dans une case mémoire de VRAM dont l'adresse est supérieure à 03FFFH, utilisez la routine NWRVRM.

00050H SETRD SET address for ReaD

Nom :	SETRD
Adresse :	00050H/ROM
Type :	MSX1 - 1 -
Rôle :	transfert dans le VDP de l'adresse de la case mémoire où doit s'effectuer la lecture
Entrée :	HL - adresse de lecture (00000H - 03FFFFH)
Sortie :	rien
Modifie :	AF
Note :	dans le cas d'une adresse supérieure à 03FFFH, utilisez la routine NSTRD.

00053H SETWRT SET address for WRiTe

Nom :	SETWRT
Adresse :	00053H/ROM
Type :	MSX1 - 1 -
Rôle :	transfert dans le VDP de l'adresse de la case mémoire où doit s'effectuer l'écriture
Entrée :	HL - adresse d'écriture (00000H - 03FFFFH)
Sortie :	rien
Modifie :	AF
Note :	dans le cas d'une adresse supérieure à 03FFFH, utilisez la routine NSTWRT.

00056H FILVRM FILl VRaM

Nom :	FILLVRM
Adresse :	00056H/ROM
Type :	MSX1 - 4 -
Rôle :	remplissage de la mémoire vidéo
Entrée :	HL - adresse de début (00000H - 03FFFFH)
	BC - longueur
	A - Donnée
Sortie :	rien
Modifie :	AF, BC
Note :	pour accéder aux adresses supérieures à 03FFFFH, utilisez la routine BIGFIL.

00059H **LDIRMV** LoaD Increment Repeat Memory with Vram

Nom :	LDIRMV
Adresse :	00059H/ROM
Type :	MSX1 - 4 -
Rôle :	transfert de la mémoire vidéo vers la mémoire centrale.
Entrée :	HL - adresse de départ en VRAM (00000H - 0FFFFH) DE - adr. destination en RAM centrale (00000H - 0FFFFH) BC - longueur du bloc à transférer
Sortie :	rien
Modifie :	tout

0005CH **LDIRVM** LoaD Increment Repeat Vram with Memory

Nom :	LDIRMV
Adresse :	0005CH/ROM
Type :	MSX1 - 4 -
Rôle :	transfert de la mémoire centrale vers la mémoire vidéo.
Entrée :	HL - adresse de départ en RAM centrale (00000H - 0FFFFH) DE - adr. destination en VRAM (00000H - 0FFFFH) BC - longueur du bloc à transférer
Sortie :	rien
Modifie :	tout

0005FH **CHGMOD** CHange MODe

Nom :	CHGMOD
Adresse :	0005FH/ROM
Type :	MSX1 - 3 -
Rôle :	changement de mode d'écran (SCREEN X).
Entrée :	A - mode désiré (0-8) SCRMOD - idem ci-dessus
Sortie :	rien
Modifie :	tout
Note :	La palette n'est pas initialisée par cette routine, appelez la routine CHGMDP dans la Sub-ROM si vous avez besoin de l'initialisation de la palette.

00062H **CHGCLR** CHanGe CoLoR

Nom :	CHGCLR
Adresse :	00062H/ROM
Type :	MSX1 - 1 -
Rôle :	changement des couleurs à l'écran
Entrée :	FORCLR - couleur de texte BAKCLR - couleur de fond BDRCLR - couleur de marge
Sortie :	rien
Modifie :	tout
Note :	Cette routine n'est rétroactive qu'en mode texte.

00066H **NMI** Non Maskable Interrupt

Nom :	NMI
Adresse :	00066H/ROM
Type :	MSX1 - 1 -
Rôle :	Procédure d'interruption non-masquable.
Entrée :	rien
Sortie :	rien
Modifie :	rien
Note :	appel au hook H.NMI

00069H **CLRSPR** CLeaR SPRites

Nom :	CLRSPR
Adresse :	00069H/ROM
Type :	MSX1 - 3 -
Rôle :	initialisation des sprites.
Entrée :	rien
Sortie :	rien
Modifie :	tout
Note :	La table des formes de sprites est effacée, les numéros de sprites sont mis aux numéros de plan, la couleur des sprites est mise à celle de la couleur du texte (FORCLR) et la position verticale des sprites est réglée à 209 pour les écrans 1 à 3, à 217 pour les screen 4 à 8.

0006CH **INITXT** INItialize TeXT mode

Nom : INITXT

Adresse : 0006CH/ROM

Type : MSX1 - 3 -

Rôle : initialisation du mode texte 40x24 (SCREEN 0).

Entrée : TXTNAM - adresse de la table des noms
TXTCGP - adresse de la table des formes

Sortie : rien

Modifie : tout

Note : Il n'est pas obligatoire de définir TXTNAM et TXTCGP car le MSX initialise leur valeur à l'allumage.

La palette n'est pas initialisée par cette routine. Si vous désirez initialiser la palette, appelez la routine INIPLT dans la Sub-ROM immédiatement après celle-ci.

0006FH **INIT32** INItialize Text 32 mode

Nom : INIT32

Adresse : 0006FH/ROM

Type : MSX1 - 3 -

Rôle : initialisation du mode texte 32x24 (SCREEN 1).

Entrée : T32NAM - adresse de la table des noms
T32CGP - adresse de la table des formes
T32COL - table des couleurs
T32ATR - table des attributs de sprites
T32PAT - table des formes de sprites

Sortie : rien

Modifie : tout

Note : Il n'est pas obligatoire de définir T32NAM, etc car le MSX initialise leur valeur à l'allumage.

La palette n'est pas initialisée par cette routine. Si vous désirez initialiser la palette, appelez la routine INIPLT dans la Sub-ROM immédiatement après celle-ci.

00072H **INIGRP** INItialize GRaPhic mode

Nom : INIGRP

Adresse : 00072H/ROM

Type : MSX1 - 3 -

Rôle : initialisation du mode graphique 256x192 (SCREEN 2).

Entrée :
GRPNAM - adresse de la table des noms
GRPCGP - adresse de la table des formes
GRPCOL - table des couleurs
GRPATR - table des attributs de sprites
GRPPAT - table des formes de sprites

Sortie : rien

Modifie : tout

Note : Il n'est pas obligatoire de définir GRPNAM, etc car le MSX initialise leur valeur à l'allumage.

La palette n'est pas initialisée par cette routine. Si vous désirez initialiser la palette, appelez la routine INIPLT dans la Sub-ROM immédiatement après celle-ci.

00075H **INIMLT** INItialize MuLTicolor mode

Nom : INIMLT

Adresse : 00075H/ROM

Type : MSX1 - 3 -

Rôle : initialisation du mode graphique « multicolore » (SCREEN 3).

Entrée :
MLTNAM - adresse de la table des noms
MLTCGP - adresse de la table des formes
MLTCOL - table des couleurs
MLTATR - table des attributs de sprites
MLTPAT - table des formes de sprites

Sortie : rien

Modifie : tout

Note : Il n'est pas obligatoire de définir MLTNAM, etc car le MSX initialise leur valeur à l'allumage.

La palette n'est pas initialisée par cette routine. Si vous désirez initialiser la palette, appelez la routine INIPLT dans la Sub-ROM immédiatement après celle-ci.

00078H **SETTXT** SET TeXT mode

Nom : SETTXT

Adresse : 00078H/ROM

Type : MSX1 - 3 -

Rôle : passage direct en mode texte 40x24

Entrée : TXTNAM - adresse de la table des noms
TXTCGP - adresse de la table des formes

Sortie : rien

Modifie : tout

Note : Il n'est pas obligatoire de définir TXTNAM et TXTCGP car le MSX initialise leur valeur à l'allumage.

La palette n'est pas initialisée par cette routine. Si vous désirez initialiser la palette, appelez la routine INIPLT dans la Sub-ROM immédiatement après celle-ci.

0007BH **SETT32** SET Text 32 mode

Nom : SETT32

Adresse : 0007BH/ROM

Type : MSX1 - 3 -

Rôle : passage direct en mode texte 32x24.

Entrée : T32NAM - adresse de la table des noms
T32CGP - adresse de la table des formes
T32COL - table des couleurs
T32ATR - table des attributs de sprites
T32PAT - table des formes de sprites

Sortie : rien

Modifie : tout

Note : Il n'est pas obligatoire de définir T32NAM, etc car le MSX initialise leur valeur à l'allumage.

La palette n'est pas initialisée par cette routine. Si vous désirez initialiser la palette, appelez la routine INIPLT dans la Sub-ROM immédiatement après celle-ci.

0007EH SETGRP SET GRaPhic mode

Nom :	SETGRP
Adresse :	0007EH/ROM
Type :	MSX1 - 3 -
Rôle :	passage direct en mode graphique 256x192.
Entrée :	GRPNAM - adresse de la table des noms GRPCGP - adresse de la table des formes GRPCOL - table des couleurs GRPATR - table des attributs de sprites GRPPAT - table des formes de sprites
Sortie :	rien
Modifie :	tout
Note :	Il n'est pas obligatoire de définir GRPNAM, etc car le MSX initialise leur valeur à l'allumage. La palette n'est pas initialisée par cette routine. Si vous désirez initialiser la palette, appelez la routine INIPLT dans la Sub-ROM immédiatement après celle-ci.

00081H SETMLT SET MuLTicolor mode

Nom :	SETMLT
Adresse :	00081H/ROM
Type :	MSX1 - 3 -
Rôle :	passage direct en mode graphique « multicolore ».
Entrée :	MLTNAM - adresse de la table des noms MLTCGP - adresse de la table des formes MLTCOL - table des couleurs MLTATR - table des attributs de sprites MLTPAT - table des formes de sprites
Sortie :	rien
Modifie :	tout
Note :	Il n'est pas obligatoire de définir MLTNAM, etc car le MSX initialise leur valeur à l'allumage. La palette n'est pas initialisée par cette routine. Si vous désirez initialiser la palette, appelez la routine INIPLT dans la Sub-ROM immédiatement après celle-ci.

00084H **CALPAT** CALculate PATtern table address

Nom :	CALPAT
Adresse :	00084H/ROM
Type :	MSX1 - 1 -
Rôle :	calcul de l'adresse du début des formes d'un sprite.
Entrée :	A - numéro du plan du sprite
Sortie :	HL - adresse cherchée
Modifie :	AF, DE, HL

00087H **CALATR** CALculate ATtRibute table address

Nom :	CALATR
Adresse :	00087H/ROM
Type :	MSX1 - 1 -
Rôle :	calcul de l'adresse du début des attributs d'un sprite.
Entrée :	A - numéro du plan du sprite
Sortie :	HL - adresse cherchée
Modifie :	AF, DE, HL

0008AH **GSPSIZ** Get SPrite SIZe

Nom :	GSPSIZ
Adresse :	0008AH/ROM
Type :	MSX1 - 1 -
Rôle :	calcul de la taille des sprites.
Entrée :	rien
Sortie :	A - nombre d'octets pour un sprite (8 ou 32)
Modifie :	AF
Note :	L'indicateur carry est positionné en cas de mode 16x16.

0008DH **GRPPRT** GRaPhic PRinT

Nom : GRPPRT

Adresse : 0008DH/ROM

Type : MSX1 - 2 -

Rôle : affichage d'un caractère sur écran graphique au pixel actuel.

Entrée : A - code ASCII du caractère à afficher

LOGOPR - code d'opération logique (pour screens 5 à 8 uniquement)

Sortie : rien

Modifie : rien

Note : Appel à la Sub-ROM si le screen est supérieur ou égal à 5.

00090H **GICINI** GI sound Chip INItialize

Nom : GICINI

Adresse : 00090H/ROM

Type : MSX1 - 1 -

Rôle : Initialisation du PSG et de données pour l'instruction PLAY.

Entrée : rien

Sortie : rien

Modifie : rien

Note : Les interruptions doivent être interdites avant l'appel de cette routine.

00093H **WRTPSG** WRiTe PSG

Nom : WRTPSG

Adresse : 00093H/ROM

Type : MSX1 - 1 -

Rôle : écriture dans un registre du PSG (« Programmable Sound Generator »)

Entrée : A - numéro de registre (0-13)

E - donnée à écrire

Sortie : rien

Modifie : rien

Note : voici le détail des registres :

R1 et R0 - fréquence émise sur la voie n°1 (12 bits). Les bits b3 à b0 de R1 représentent les quatre bits de poids fort de la fréquence, R0 contient les huits bits de poids faible de la fréquence.

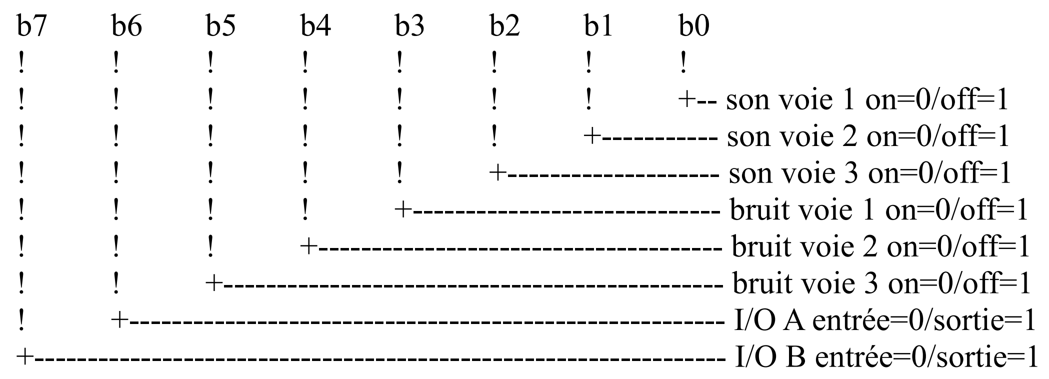
R3 et R2 - fréquence émise sur la voie n°2 (12 bits). Voir R1 et R0 pour les

explications.

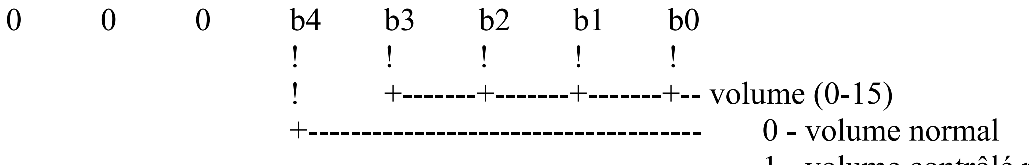
R4 et R5 - fréquence émise sur la voie n°3 (12 bits). Voir R1 et R0 pour les explications.

R6 - fréquence du générateur de bruit blanc.

R7 - contrôleur de mélange des voies :



R8 - volume de la voie n°1



R9 - volume de la voie n°2 (voir R8)

R10 - volume de la voie n°3 (voir R8)

R12 et R11 - fréquence de l'enveloppe

R13 - forme de l'enveloppe (0, 4, 8, 10, 11, 12, 13 ou 14)

- pour arrêter complètement le son sur une voie, il faut mettre le volume de voie à 0 ET modifier le registre de contrôle (R7).
- afin de maintenir la compatibilité, il ne faut en aucun cas utiliser les fonctions d'entrée/sortie du PSG.

00096H **RDPSG** ReaD PSG

Nom :	RDPSG
Adresse :	00096H/ROM
Type :	MSX1 - 1 -
Rôle :	lecture d'un registre du PSG
Entrée :	A - numéro du registre
Sortie :	A - contenu du registre
Modifie :	rien
Note :	voir la routine WRTPSG pour le détail des registres du PSG.

00099H **STRTMS** STarT background MuSic task

Nom :	STRTMS
Adresse :	00099H/ROM
Type :	MSX1 - 1 -
Rôle :	démarre le travail de fond pour PLAY
Entrée :	rien
Sortie :	rien
Modifie :	tout

0009CH **CHSNS**

Nom :	CHSNS
Adresse :	0009CH/ROM
Type :	MSX1 - 1 -
Rôle :	mise à jour de l'affichage des touches de fonction d'une part, et vérification du buffer clavier d'autre part.
Entrée :	rien
Sortie :	l'indicateur zéro à 0 - il y a un caractère dans le buffer clavier
Modifie :	AF

0009FH **CHGET** CHaracter GET

Nom :	CHGET
Adresse :	0009FH/ROM
Type :	MSX1 - 1 -
Rôle :	attente d'un caractère au clavier et retour avec son code
Entrée :	rien
Sortie :	A - code ASCII du caractère
Modifie :	AF
Note :	appel au hook H.CHGE

000A2H	**CHPUT**	CHaracter outPUT
	Nom :	CHPUT
	Adresse :	000A2H/ROM
	Type :	MSX1 - 1 -
	Rôle :	sortie d'un caractère sur écran en mode texte
	Entrée :	A - code du caractère
	Sortie :	rien
	Modifie :	rien
	Note :	appel au hook H. CHPU

000A5H	**LPTOUT**	Line PrinTer OUT
	Nom :	LPTOUT
	Adresse :	000A5H/ROM
	Type :	MSX1 - 1 -
	Rôle :	sortie d'un caractère sur imprimante
	Entrée :	A - code du caractère
	Sortie :	indicateur carry à 1 - la routine a été interrompue par un CTRL+STOP
	Modifie :	F
	Note :	appel au hook H.LPTO Le CTRL+STOP permet de revenir de cette routine en cas de problème avec l'imprimante.

000A8H	**LPTSTT**	Line PrinTer STaTus
	Nom :	LPTSTT
	Adresse :	000A8H/ROM
	Type :	MSX - 1 -
	Rôle :	vérification de l'état de l'imprimante
	Entrée :	rien
	Sortie :	A - 255 si l'imprimante est prête, 0 si elle ne l'est pas indicateur zéro - 0 si l'imprimante est prête, 1 si elle ne l'est pas
	Modifie :	AF
	Note :	Appel au hook H.LPTS

000ABH **CNVCHR** CoNVert CHaRacter

Nom :	CNVCHR
Adresse :	000ABH/ROM
Type :	MSX1 - 1 -
Rôle :	test si header graphique et transformation du caractère suivant si c'est le cas.
Entrée :	A - code du caractère
Sortie :	indicateur carry à 0 - le code reçu est un header graphique (01)
	indicateur carry à 1 - indicateur zéro à 1 - le code a été transformé
	indicateur carry à 1 - indicateur zéro à 0 - le code n'a pas été transformé car ce n'était pas un caractère graphique
Modifie :	AF
Note :	en entrée, si A contient 01 (code de header graphique), le flag GRPHED est mis à 1. Le caractère suivant envoyé à la routine sera transformé (masquage du bit 6) à condition que son code soit compris entre 65 et 95.

000AEH **PINLIN** Program INput LINe

Nom :	PINLIN
Adresse :	000AEH/ROM
Type :	MSX1 - 1 -
Rôle :	prise de ligne au clavier jusqu'à un STOP ou un retour chariot
Entrée :	rien
Sortie :	HL - adresse du premier octet dans le buffer -1
	indicateur carry à 1 - un ^C a eu lieu
Modifie :	tout
Note :	appel au hook H.PINL
	si AUTFLG est à 1, appel à INLIN

000B1H **INLIN** INput LINe

Nom :	INLIN
Adresse :	000B1H/ROM
Type :	MSX1 - 1 -
Rôle :	prise de ligne au claiver jusqu'à un STOP ou un retour chariot
Entrée :	rien
Sortie :	HL - adresse du premier octet dans le buffer -1
	indicateur carry à 1 - un ^C a eu lieu
Modifie :	tout
Note :	appel au hook H.INLI
	par rapport à PINLIN, cette routine termine la ligne précédente (voir LINTTB)

000B4H **QINLIN** Questiuon mark and INput LINe

Nom :	QINLIN
Adresse :	000B4H/ROM
Type :	MSX1 - 1 -
Rôle :	idem INLIN sauf que cette routine affiche un point d'interrogation.
Entrée :	rien
Sortie :	HL - adresse du premier octet dans le buffer -1
	indicateur carry à 1 - un ^C a eu lieu
Modifie :	tout
Note :	appel au hook H.QINL

000B7H **BREAKX** BREAK X

Nom :	BREAKX
Adresse :	000B7H/ROM
Type :	MSX1 - 1 -
Rôle :	Test du CTRL+STOP
Entrée :	rien
Sortie :	indicateur carry à 1 - ^C en cours
Modifie :	AF
Note :	Exemple d'utilisation :

```
LOOP :    CALL  BREAKX
          JR    NC, LOOP
          RET
```

000BAH **ISCNTC** IS CoNTrol C ?

Nom : ISCNTC

Adresse : 000BAH/ROM

Type : MSX1 - 1 -

Rôle : test de la touche STOP (éventuellement avec CTRL) pour l'interpréteur Basic

Entrée : BASROM - 1 si programme en cartouche (STOP interdit)

Sortie : rien

Modifie : rien

Note : réservé au Basic

000BDH **CKCNTC** ChecK CoNtrol C

Nom : CKCNTC

Adresse : 000BDH/ROM

Type : MSX1 - 1 -

Rôle : test de la touche STOP (éventuellement avec CTRL) pour l'interpréteur Basic

Entrée : BASROM - 1 si programme en cartouche (STOP interdit)

Sortie : rien

Modifie : rien

Note : réservé au Basic, par rapport à ISCNTC, cette routine met le pointeur de texte du Basic à 0 (après sauvegarde) afin d'interdire un CONT (« CONTinue »).

000C0H **BEEP** BEEP buzzer

Nom : BEEP

Adresse : 000C0H/ROM

Type : MSX1 - 3 -

Rôle : émission d'un bref bip sonore

Entrée : rien

Sortie : rien

Modifie : tout

000C3H **CLS** CLear Screen

Nom :	CLS
Adresse :	000C3H/ROM
Type :	MSX1 - 3 -
Rôle :	effacement de l'écran dans tous les modes
Entrée :	indicateur zéro à 1 - obligatoire
Sortie :	rien
Modifie :	AF, BC

000C6H **POSIT** POSITion cursor

Nom :	POSIT
Adresse :	000C6H/ROM
Type :	MSX1 - 1 -
Rôle :	modification des coordonnées du curseur
Entrée :	H - numéro de colonne
	L - numéro de ligne
Sortie :	rien
Modifie :	AF
Note :	Bien que n'ayant pas été modifiée sur MSX2, cette routine fonctionne parfaitement en mode 80 colonnes.

000C9H **FNKSB** FuNction Key diSplay enaBled

Nom :	FNKSB
Adresse :	000C9H/ROM
Type :	MSX1 - 1 -
Rôle :	vérification de l'indicateur, et affichage des touches de fonction si nécessaire.
Entrée :	CNSDFG - 1 affichage des touches de fonction
	0 ne fait rien
Sortie :	rien
Modifie :	tout

000CCH **ERAFNK** ERAse FuNction Key

Nom :	ERAFNK
Adresse :	000CCH/ROM
Type:	MSX1 - 1 -
Rôle :	effacement de l'affichage des touches de fonction
Entrée :	rien
Sortie :	rien
Modifie :	tout
Note :	appel au hook H.ERAF

000CFH **DSPFNK** DiSPlay FuNction Key

Nom :	DSPFNK
Adresse :	000CFH/ROM
Type:	MSX1 - 2 -
Rôle :	affichage des touches de fonction
Entrée :	rien
Sortie :	rien
Modifie :	tout
Note :	appel au hook H.DSPF

000D2H **TOTEXT** force screen TO TEXT mode

Nom :	TOTEXT
Adresse :	000D2H/ROM
Type:	MSX1 - 1 -
Rôle :	passage de mode graphique en mode texte
Entrée :	OLDMOD - mode texte dans lequel nous étions avant le passage en mode graphique. Nous retournons dans ce mode
Sortie :	rien
Modifie :	tout
Note :	appel au hook H.TOTE
	Cette routine n'est pas modifiée sur MSX2 mais elle appelle CHGMDP et passe donc par la Sub-ROM

000D5H **GTSTCK** GeT joySTiCK status

Nom : GTSTCK

Adresse : 000D5H/ROM

Type: MSX1 - 1 -

Rôle : renvoi de la direction actuelle du joystick

Entrée : A - numéro du joystick (0-3)

Sortie : A - direction du joystick

Modifie : tout

Note : le joystick numéro 0 correspond au clavier (les quatre flèches)

la direction du joystick est un nombre de 0 à 8

```
8 1 2    Lors de l'utilisation du clavier comme joystick 0,
 \ | /   il faut appuyer simultanément sur deux flèches afin
7-0-3    d'obtenir une diagonale. Par exemple, flèche vers le bas et
 / | \   flèche vers la droite donnent 4.
6 5 4
```

000D8H **GTTRIG** GeT TRIGger button status

Nom : GTTRIG

Adresse : 000D8H/ROM

Type: MSX1 - 1 -

Rôle : renvoi de l'état du bouton de tir

Entrée : A - numéro du bouton de tir (0-4)

Sortie : A - 0FFH si le bouton n'est pas enfoncé

0 si ce n'est pas le cas

Modifie : AF

Note : le bouton de tir 0 correspond à la barre d'espacement, les boutons 1 et 2 correspondent au premier bouton de tir de chaque joystick alors que les boutons 3 et 4 correspondent au second bouton de tir des joysticks

000DBH **GTPAD** GeT touch PAD

Nom : GTPAD

Adresse : 000DBH/ROM

Type: MSX1 - 1 -

Rôle : lecture de la tablette graphique

Entrée : A - numéro de l'opération à effectuer (0-7 ou 0-19 sur MSX2)

Sortie : A - valeur résultante de l'opération

Modifie : tout

Note : la tablette graphique doit être une NEC PC-6051 ou compatible. Voici la liste des opérations possibles :

0 - la tablette graphique branchée dans le port joystick n°1 est-elle prête ? (0FFH si oui, 0 si non)

1 - coordonnée en X actuelle de la tablette branchée dans le port joystick n°1

2 - coordonnée en Y actuelle de la tablette branchée dans le port joystick n°1

3 - le crayon est-il sur la tablette branchée dans le port joystick n°1 ? (0FFH si oui, 0 si non)

4 - la tablette graphique branchée dans le port joystick n°2 est-elle prête ? (0FFH si oui, 0 si non)

5 - coordonnée en X actuelle de la tablette branchée dans le port joystick n°2

6 - coordonnée en Y actuelle de la tablette branchée dans le port joystick n°2

7 - le crayon est-il sur la tablette branchée dans le port joystick n°2 ? (0FFH si oui, 0 si non)

Pour plus de précisions sur l'extension de cette routine à la souris, au track-ball et au crayon optique, voir la routine NEWPAD (001ADH en Sub-ROM)

000DEH **GTPDL** GeT PaDdLe

Nom : GTPDL

Adresse : 000DEH/ROM

Type : MSX1 - 2 -

Rôle : lecture du paddle

Entrée : A - paramètre d'entrée (1-12)

Sortie : A - valeur du paddle

Modifie : tout

Note : Voici la liste des paramètres d'entrée :

1 - paddle A dans le port joystick 1

2 -	paddle A dans le port joystick 2
3 -	paddle B dans le port joystick 1
4 -	paddle B dans le port joystick 2
5 -	paddle C dans le port joystick 1
6 -	paddle C dans le port joystick 2
7 -	paddle D dans le port joystick 1
8 -	paddle D dans le port joystick 2
9 -	paddle E dans le port joystick 1
10 -	paddle E dans le port joystick 2
11 -	paddle F dans le port joystick 1
12 -	paddle F dans le port joystick 2

000E1H **TAPION** TAPe In ON

Nom : TAPION

Adresse : 000E1H/ROM

Type : MSX1 - 1 -

Rôle : démarrage du moteur du magnétophone et lecture du header

Entrée : rien

Sortie : indicateur carry à 1 - opération interrompue (après une erreur par exemple)

Modifie : tout

Note : Il existe deux types de headers, les longs et les courts. Les premiers servent à marquer les débuts de fichiers alors que les seconds séparent les différentes parties de fichier. Voir TAPOON.

La vitesse en bauds (1200 ou 2400) est déterminée lors de la lecture du header.

000E4H **TAPIN** TAPe IN

Nom : TAPIN

Adresse : 000E4H/ROM

Type : MSX1 - 1 -

Rôle : Lecture d'un octet depuis la cassette

Entrée : rien

Sortie : A - octet lu

indicateur carry à 1 - opération interrompue

Modifie : tout

000E7H **TAPIOF** TAPe In OFf

Nom :	TAPIOF
Adresse :	000E7H/ROM
Type :	MSX1 - 1 -
Rôle :	fin de lecture, arrêt moteur
Entrée :	rien
Sortie :	rien
Modifie :	rien

000EAH **TAPOON** TAPe Out ON

Nom :	TAPOON
Adresse :	000EAH/ROM
Type :	MSX1 - 1 -
Rôle :	Démarrage du moteur du magnétophone et écriture d'un header
Entrée :	A - 0 pour un header court, 1 pour un header long
Sortie :	indicateur carry à 1 - opération interrompue (après un CTRL+STOP par exemple)
Modifie :	tout
Note :	Il existe deux types de headers, les longs et les courts. Les premiers servent à marquer les débuts de fichiers alors que les seconds séparent les différentes parties de fichier

Trois types de fichiers cassettes sont disponibles sur MSX : fichiers Basic, fichiers ASCII et fichiers binaires. Voici le détail de chacun d'eux :

fichier Basic :	header long - 10 x 0D3H - nom du programme Basic (6 octets)
puis	header court - programme Basic condensé et 7 x 000H
fich. ASCII :	header long - 10 x 0EAH - nom du fichier (6 octets)
puis	header court - 256 x 1 octet de données
	header court - 256 x 1 octet de données
	idem jusqu'à la fin du fichier...
fichier bin :	header long - 10 x 0D0H - nom du programme (6 octets)
puis	header court - adresse de chargement (2 octets) - adresse de fin (2 octets) - adresse de début d'exécution (2 octets) - programme en lui-même

000EDH **TAPOUT** TAPe OUT

Nom : TAPOUT

Adresse : 000EDH/ROM

Type : MSX1 - 1 -

Rôle : écriture d'un octet sur la cassette

Entrée : A - octet à écrire

Sortie : indicateur carry à 1 - opération interrompue

Modifie : tout

000F0H **TAPOOF** TAPe Out OFf

Nom : TAPOOF

Adresse : 000F0H/ROM

Type : MSX1 - 1 -

Rôle : fin d'écriture, arrêt moteur

Entrée : rien

Sortie : rien

Modifie : rien

000F3H **STMOTR** SeT MOTor

Nom : STMOTR

Adresse : 000F3H/ROM

Type : MSX1 - 1 -

Rôle : Changement d'état du moteur du magnétophone

Entrée : A - 000H pour arrêter le moteur

001H pour démarrer le moteur

0FFH pour inverser l'état actuel

Sortie : rien

Modifie : AF

000F6H **LFTQ** LeFT in Queue

Nom : LFTQ

Adresse : 000F6H/ROM

Type : MSX1 - 1 -

Rôle : donne le nombre d'octets restants dans une queue

Entrée : A - numéro de queue

Sortie : A - nombre d'octets

Modifie : AF, BC, HL

000F9H **PUTQ** PUT in Queue

Nom : PUTQ

Adresse : 000F9H/ROM

Type : MSX1 - 1 -

Rôle : empilage d'un octet sur une queue

Entrée : A - numéro de queue

E - donnée à mettre

Sortie : indicateur zéro à 1 - la queue est entièrement remplie

Modifie : AF, BC, HL

En SCREEN 2 à 4, pour toutes les routines graphiques qui suivent (RIGHTC à SCANL), « curseur graphique » désigne un pixel déterminé par deux variables système : CLOC et CMASK. La première donne l'adresse en mémoire vidéo correspondante au pixel qui nous intéresse alors que la seconde indique quel masque il est nécessaire d'appliquer à l'octet pour allumer juste le bon pixel.

000FCH **RIGHTC** RIGHT Current

Nom : RIGHTC

Adresse : 000FCH/ROM

Type : MSX1 - 2 -

Rôle : déplacement du curseur graphique vers la droite

Entrée : rien

Sortie : rien

Modifie : AF

000FFH **LEFTC** LEFT Current

Nom : LEFTC

Adresse : 000FFH/ROM

Type : MSX1 - 2 -

Rôle : déplacement du curseur graphique vers la gauche

Entrée : rien

Sortie : rien

Modifie : AF

00102H **UPC** UP Current

Nom : UPC

Adresse : 00102H/ROM

Type : MSX1 - 2 -

Rôle : déplacement du curseur graphique vers le haut

Entrée : rien

Sortie : rien

Modifie : AF

00105H **TUPC** Test and UP Current

Nom : TUPC

Adresse : 00105H/ROM

Type : MSX1 - 2 -

Rôle : déplacement du curseur vers le haut après vérification de la validité des coordonnées

Entrée : rien

Sortie : indicateur carry à 1 - le déplacement n'a pas eu lieu car le curseur graphique était déjà en 0

Modifie : AF

00108H **DOWNC** DOWN Current

Nom : DOWNC

Adresse : 00108H/ROM

Type : MSX1 - 2 -

Rôle : déplacement du curseur graphique vers le bas

Entrée : rien

Sortie : rien

Modifie : AF

0010BH **TDOWNC** Test and DOWN Current

Nom :	TUPC
Adresse :	0010BH/ROM
Type :	MSX1 - 2 -
Rôle :	déplacement du curseur vers le bas après vérification de la validité des coordonnées
Entrée :	rien
Sortie :	indicateur carry à 1 - le déplacement n'a pas eu lieu car le curseur graphique était déjà en 191 (ou 211)
Modifie :	AF

0010EH **SCALXY** SCALe X & Y

Nom :	SCALXY
Adresse :	0010EH/ROM
Type :	MSX1 - 2 -
Rôle :	vérification et éventuellement ajustement des valeurs du curseur graphique
Entrée :	BC - valeur de X (0-0FFFFH)
	DE - valeur de Y (0-0FFFFH)
Sortie :	BC - valeur de X valide
	DE - valeur de Y valide
	indicateur carry à 0 - une des coordonnées était hors de l'écran
Modifie :	AF
Note :	Lorsque l'une des deux coordonnées est négative, cette routine prend 0 comme nouvelle coordonnée. De même, si une coordonnée dépasse sa valeur maximale autorisée (255 ou 511 pour X, 191 ou 211 pour Y), la routine prend la valeur maximale autorisée comme nouvelle coordonnée.

00111H **MAPXYC** MAP X&Y to Current

Nom : MAPXYC

Adresse : 00111H/ROM

Type : MSX1 - 2 -

Rôle : transformation de coordonnées d'un pixel en une adresse mémoire vidéo et un masque (utile surtout pour MSX1)

Entrée : BC - coordonnée horizontale du pixel à allumer

DE - coordonnée verticale du pixel à allumer

Sortie : Dans les SCREEN 2 à 4 :

CLOC - adresse en VRAM de l'octet à modifier

CMASK - masque à appliquer à l'octet ci-dessus pour allumer le pixel qui nous intéresse

Dans les SCREEN 5 à 8

CLOC - coordonnée horizontale du pixel

CMASK - coordonnée verticale du pixel

Modifie : F

00114H **FETCHC** FETCH Current

Nom : FETCHC

Adresse : 00114H/ROM

Type : MSX1 - 1 -

Rôle : récupération de l'adresse VRAM actuelle et du masque actuel (ou coordonnées pour les SCREEN 5 à 8)

Entrée : rien

Sortie : HL - adresse VRAM du curseur graphique (SCREEN 2 à 4) ou coordonnée horizontale (SCREEN 5 à 8)

A - masque du curseur graphique (SCREEN 2 à 4) ou coordonnée verticale (SCREEN 5 à 8)

Modifie : F

00117H **STOREC** STORE Current

Nom :	STOREC
Adresse :	00117H/ROM
Type :	MSX1 - 1 -
Rôle :	sauvegarde de l'adresse VRAM actuelle et du masque actuel (ou des coordonnées)
Entrée :	HL - adresse VRAM du curseur graphique (ou coordonnée horizontale)
	A - masque du curseur graphique (ou coordonnée verticale)
Sortie :	rien
Modifie :	rien

0011AH **SETATR** SET ATtRibute byte

Nom :	SETATR
Adresse :	0011AH/ROM
Type :	MSX1 - 4 -
Rôle :	modification de l'octet d'attribut (ATRBYT)
Entrée :	A - valeur du nouvel attribut
Sortie :	indicateur carry à 1 - attribut non valide (>15)
Modifie :	F
Note :	Cette routine n'est valable que dans les SCREEN 0 à 4

0011DH **READC** READ Current

Nom :	READC
Adresse :	0011DH/ROM
Type :	MSX1 - 2 -
Rôle :	lecture de l'attribut du pixel actuel
Entrée :	CLOC et CMASK - coordonnées du pixel actuel
Sortie :	A - attribut du pixel
Modifie :	F
Note :	On appelle « pixel actuel » le pixel déterminé par le curseur graphique

00120H **SETC** SET Current

Nom : SETC

Adresse : 00120H/ROM

Type : MSX1 - 2 -

Rôle : allumage du pixel actuel à l'écran

Entrée : CLOC - adresse mémoire vidéo à modifier (SCREEN à 4) ou coordonnée horizontale (SCREEN 5 à 8)

CMASK - masque à appliquer (SCREEN 2 à 4) ou coordonnée verticale (SCREEN 5 à 8)

ATRBYT - couleur du pixel (0-15 pour les SCREEN 2 à 7, 0 à 0FFH pour le SCREEN 8)

Sortie : rien

Modifie : AF

00123H **NSETCX** Next SET Current X

Nom : NSETCX

Adresse : 00123H/ROM

Type : MSX1 - 1 -

Rôle : affichage de plusieurs pixels vers la droite

Entrée : HL - nombre de pixels à allumer

CLOC et CMASK - curseur graphique

ATRBYT - couleur du pixel (0-15 pour les SCREEN 2 à 7, 0 à 0FFH pour le SCREEN 8)

Sortie : rien

Modifie : tout

00126H **GTASPC** GeT ASPeCt ratio

Nom : GTASPC

Adresse : 00126H/ROM

Type : MSX1 - 1 -

Rôle : renvoi de l'aspect du cercle

Entrée : rien

Sortie : DE - contenu de ASPCT1

HL - contenu de ASPCT2

Modifie : rien

00129H **PNTINI** PaiNT INItialize

Nom : PNTINI

Adresse : 00129H/ROM

Type : MSX1 - 1 -

Rôle : initialisation de la procédure de peinture

Entrée : rien

Sortie : rien

Modifie : AF

0012CH **SCANR** SCAN Right

Nom : SCANR

Adresse : 0012CH/ROM

Type : MSX1 - 2 -

Rôle : recherche vers la droite du premier pixel d'une autre couleur

Entrée : DE - nombre de pixels sur lesquels la recherche doit s'effectuer

Sortie : DE - position du pixel recherché

Modifie : tout

Note : Un exemple, si vous chargez DE avec 100H et que le premier pixel d'une autre couleur se trouve à 4 pixels, la routine renverra 0FCH dans DE

0012FH **SCANL** SCAN Left

Nom : SCANL

Adresse : 0012FH/ROM

Type : MSX1 - 2 -

Rôle : recherche vers la gauche du premier pixel d'une autre couleur

Entrée : DE - nombre de pixels sur lesquels la recherche doit s'effectuer

Sortie : DE - position du pixel recherché

Modifie : tout

Note : Un exemple, si vous chargez DE avec 120H et que le premier pixel d'une autre couleur se trouve à 9 pixels, la routine renverra 117H dans DE

00132H **CHGCAP** CHanGe CAPs lamp status

Nom : CHGCAP

Adresse : 00132H/ROM

Type : MSX1 - 1 -

Rôle : allume ou éteint le voyant CAPS LOCK

Entrée : A - 0 pour éteindre le voyant
1 pour allumer le voyant

Sortie : rien

Modifie : AF

00135H **CHGSND** CHanGe SouND port status

Nom : CHGSND

Adresse : 00135H/ROM

Type : MSX1 - 1 -

Rôle : allume ou éteint le son

Entrée : A - 0 pour éteindre
1 pour allumer

Sortie : rien

Modifie : AF

00138H **RSLREG** Read primary SLot REGister

Nom : RSLREG

Adresse : 00138H/ROM

Type : MSX1 - 1 -

Rôle : lecture du registre des slots primaires

Entrée : rien

Sortie : A - valeur du registre

Modifie : rien

Note : décodage du registre :

```
b7      b6      b5      b4      b3      b2      b1      b0
!       !       !       !       !       !       !       !
!       !       !       !       !       !       +-------+-- slot pour la page 0
!       !       !       !       +-------+------------------ slot pour la page 1
!       !       +-------+---------------------------------- slot pour la page 2
+-------+-------------------------------------------------- slot pour la page 3
```

0013BH **WSLREG** Write SLot REGister

Nom : WSLREG

Adresse : 0013BH/ROM

Type : MSX1 - 1 -

Rôle : écriture dans le registre des slots primaires

Entrée : A - donnée à écrire

Sortie : rien

Modifie : rien

Note : décodage du registre :

```
b7      b6      b5      b4      b3      b2      b1      b0

!       !       !       !       !       !       !       !
!       !       !       !       !       !       +-------+-- slot pour la page 0
!       !       !       !       +-------+------------------- slot pour la page 1
!       !       +-------+----------------------------------- slot pour la page 2
+-------+--------------------------------------------------- slot pour la page 3
```

0013EH **RDVDP** ReaD VDP status register

Nom : RDVDP

Adresse : 0013EH/ROM

Type : MSX1 - 1 -

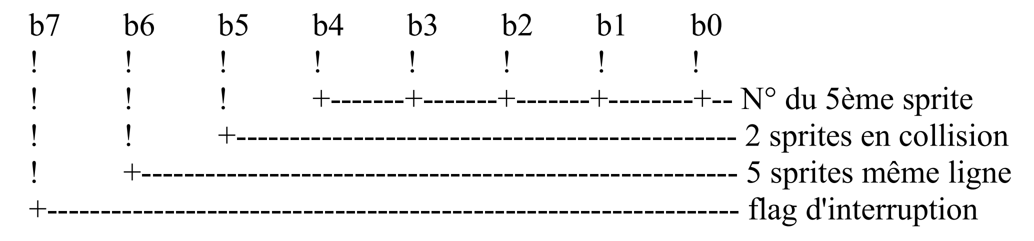
Rôle : lecture du registre d'état du VDP

Entrée : rien

Sortie : A - contenu du registre

Modifie : rien

Note : Voici le détail du contenu du registre d'état du processeur vidéo (voir le chapitre sur le processeur vidéo pour plus de précisions sur ce registre)



00141H **SNSMAT** ScaN Specified row in keyboar MATrix

Nom : SNSMAT

Adresse : 00141H/ROM

Type : MSX1 - 1 -

Rôle : lecture d'une ligne dans la matrice clavier

Entrée : A - numéro de la ligne (0-8 ou 0-10 pour les claviers avec pavé numérique)

Sortie : A - état de la ligne, les bits correspondants aux touches enfoncées sont à 0

Modifie : AF, C

Note : voici le détail de la matrice clavier (clavier AZERTY)

	b7	b6	b5	b4	b3	b2	b1	b0
L00 :	è	§	(	'	«	é	&	à
L01 :	m	$	^	<	-	)	ç	!
L02 :	b	q	DEAD	=	:	;	#	ù
L03 :	j	i	h	g	f	e	d	c
L04 :	r	a	p	o	n	,	l	k
L05 :	w	y	x	z	v	u	t	s
L06 :	F3	F2	F1	CODE	CAP	GRAPH	CTRL	SHIFT
L07 :	RET	SELECT	BS	STOP	TAB	ESC	F5	F4
L08 :	droite	bas	haut	gauche	SUP	INS	EFF	SPACE
L09 :	4	3	2	1	0	opt	opt	opt
L10 :	.	,	-	9	8	7	6	5

Les deux dernières lignes ne sont utilisées que sur les claviers possédant des pavés numériques. Les touches opt (option) sont laissées à la discrétion du constructeur. La touche DEAD n'existe pas sur certains claviers, sur d'autres, elle ne porte aucune inscription.

00144H **PHYDIO** PHYsical Disk I/O

Nom : PHYDIO

Adresse : 00144H/ROM

Type : MSX1 - 1 -

Rôle : rien sauf appel au hook

Entrée : rien

Sortie : rien

Modifie : rien

Note : appel au hook H.PHYD

00147H **FORMAT** disk FORMATter

Nom :	FORMAT
Adresse :	00147H/ROM
Type :	MSX1 - 1 -
Rôle :	rien sauf appel au hook
Entrée :	rien
Sortie :	rien
Modifie :	rien
Note :	appel au hook H.FORM

0014AH **ISFLIO** IS FiLe I/O

Nom :	ISFLIO
Adresse :	0014AH/ROM
Type :	MSX1 - 1 -
Rôle :	vérification des entrées/sorties
Entrée :	rien
Sortie :	indicateur zéro à 0 - entrée/sortie en cours
Modifie :	AF
Note :	appel au hook H.ISFL

0014DH **OUTDLP** OUT Do Line Printer

Nom :	OUTDLP
Adresse :	0014DH/ROM
Type :	MSX1 - 1 -
Rôle :	sortie d'un caractère sur imprimante
Entrée :	A - code ASCII du caractère à sortir
Sortie :	rien
Modifie :	F
Note :	Par rapport à LPTOUT, cette routine transforme les TAB en espaces, transforme les caractères graphiques lors de l'utilisation d'une imprimante non-MSX, et saute à « device I/O error » lors d'un CTRL+STOP.

00150H **GETVCP** GET VoiCe Pointer

Nom :	GETVCP
Adresse :	00150H/ROM
Type :	MSX1 - 1 -
Rôle :	mise en place du pointeur pour une queue musicale
Entrée :	A - numéro de la voie
Sortie :	HL - pointeur désiré
Modifie :	AF

00153H **GETVC2** GET VoiCe 2

Nom :	GETVC2
Adresse :	00153H/ROM
Type :	MSX1 - 1 -
Rôle :	Déplace le pointeur pour VOICEN
Entrée :	L - déplacement désiré
Sortie :	HL - nouvelle valeur du pointeur
Modifie :	AF

00156H **KILBUF** KILl BUFfer

Nom :	KILBUF
Adresse :	00156H/ROM
Type :	MSX1 - 1 -
Rôle :	destruction du buffer clavier
Entrée :	rien
Sortie :	rien
Modifie :	HL

00159H **CALBAS** CALl BASic

Nom :	CALBAS
Adresse :	00159H/ROM
Type :	MSX1 - 1 -
Rôle :	appel à des sous-programmes dans la ROM Basic depuis un autre slot (appel inter-slot)
Entrée :	IX - adresse à appeler
Sortie :	rien
Modifie :	AF, IX, IY, registres auxiliaires

0015CH **SUBROM** call SUBROM

Nom :	SUBROM
Adresse :	0015CH/ROM
Type :	MSX2
Rôle :	appel à une adresse dans la Sub-ROM
Entrée :	IX - adresse à appeler
Sortie :	rien
Modifie :	IY, registres auxiliaires du Z80
Note :	méthode pour utiliser cette routine :

```
EXPL: NOP                 une fois la routine en Sub-ROM
      PUSH IX             exécutée, on aura un retour au
      LD   IX,addr        programme qui a appelé EXPL
      JP   SUBROM
```

0015FH **EXTROM** call EXTernal ROM

Nom :	EXTROM
Adresse :	0015FH/ROM
Type :	MSX2
Rôle :	appel direct à une adresse dans la Sub-ROM
Entrée :	IX - adresse à appeler
Sortie :	rien
Modifie :	IY, registres auxiliaires du Z80

00162H **CHKSLZ** CHecK SLot Z

Nom :	CHKSLZ
Adresse :	00162H/ROM
Type :	MSX2
Rôle :	recherche du slot de la Sub-ROM
Entrée :	rien
Sortie :	rien
Modifie :	tout

00165H **CHKNEW** CHecK NEW screen mode

Nom :	CHKNEW
Adresse :	00165H/ROM
Type :	MSX2
Rôle :	vérification du SCREEN
Entrée :	rien
Sortie :	indicateur carry à 0 - le SCREEN actuel est entre 5 et 8
Modifie :	AF

00168H **EOL** erase to End Of Line

Nom :	EOL
Adresse :	00168H/ROM
Type :	MSX2
Rôle :	effacement jusqu'à la fin de la ligne (équivalent au CTRL E du Basic)
Entrée :	H - colonne à partir de la quelle il faut effacer
	L - ligne à effacer
Sortie :	rien
Modifie :	tout

0016BH **BIGFIL** BIG FILl

Nom :	BIGFIL
Adresse :	0016BH/ROM
Type :	MSX2
Rôle :	remplissage de la mémoire vidéo
Entrée :	HL - adresse de début (00000H-0FFFFH)
	BC - longueur
	A - Donnée
Sortie :	rien
Modifie :	AF, BC

0016EH **NSETRD** New SET address for ReaD

Nom :	NSETRD
Adresse :	0016EH/ROM
Type :	MSX2
Rôle :	transfert dans le VDP de l'adresse de la case mémoire où doit s'effectuer la lecture
Entrée :	HL - adresse de lecture (00000H-0FFFFH)
Sortie :	rien
Modifie :	AF

00171H **NSTWRT** New SeT address for WRiTe

Nom :	NSTWRT
Adresse :	00171H/ROM
Type :	MSX2
Rôle :	transfert dans le VDP de l'adresse de la case mémoire où doit s'effectuer l'écriture
Entrée :	HL - adresse d'écriture (00000H-0FFFFH)
Sortie :	rien
Modifie :	AF

00174H **NRDVRM** New ReaD VRaM

Nom :	NRDVRM
Adresse :	00174H/ROM
Type :	MSX2
Rôle :	lecture d'une case mémoire de VRAM
Entrée :	HL - adresse de la case mémoire à lire (00000H-0FFFFH)
Sortie :	A - contenu de la case mémoire lue
Modifie :	F

00177H **NWRVRM** New WRite VRaM

Nom :	NWRVRM
Adresse :	00177H/ROM
Type :	MSX2
Rôle :	écriture dans une case mémoire de VRAM
Entrée :	HL - adresse de la case mémoire (00000H-0FFFFH)
	A - donnée à écrire
Sortie :	rien
Modifie :	AF

Voici, en plus des routines du Bios, quelques entrées dans la ROM Basic. Celles-ci sont garanties par ASCII (« How to use graphic routines of MSX » du 05/02/86).

058FCH **LINE** LINE

Nom :	LINE
Adresse :	058FCH/ROM BASIC
Type :	MSX1
Rôle :	Tracé d'une ligne
Entrée :	GXPOS - abscisse du pixel de départ
	GYPOS - ordonnée du pixel de départ
	BC - abscisse du pixel d'arrivée
	DE - ordonnée du pixel d'arrivée
Sortie :	rien
Modifie :	BC, DE, HL
Note :	Il est nécessaire d'appeler SETATR (0011AH) avant d'utiliser cette routine.
	LINE fonctionne dans les SCREEN 2 à 8.

058C1H	**BOXFIL**	BOX FILl
	Nom :	BOXFIL
	Adresse :	058C1H/ROM BASIC
	Type :	MSX1
	Rôle :	tracé d'un rectangle plein
	Entrée :	GXPOS - abscisse du pixel de départ
		GYPOS - ordonnée du pixel de départ
		BC - abscisse du pixel d'arrivée
		DE - ordonnée du pixel d'arrivée
	Sortie :	rien
	Modifie :	HL
	Note :	Il est nécessaire d'appeler SETATR (0011AH) avant d'utiliser cette routine.
		BOXFIL fonctionne dans les SCREEN 2 à 4

057F5H	**PSET**	Pixel SET
	Nom :	PSET
	Adresse :	057F5H/ROM BASIC
	Type :	MSX1
	Rôle :	allumage d'un pixel à l'écran
	Entrée :	BC - abscisse du pixel à allumer
		DE - ordonnée du pixel à allumer
	Sortie :	rien
	Modifie :	HL
	Note :	Il est nécessaire d'appeler SETATR (0011AH) avant d'utiliser cette routine.
		PSET fonctionne dans les SCREEN 2 à 8.

059E3H	PAINT	PAINT
	Nom :	PAINT
	Adresse :	059E3H/ROM BASIC
	Type :	MSX1
	Rôle :	remplissage d'une surface à l'écran
	Entrée :	BC - abscisse du pixel où commence le remplissage
		DE - ordonnée du pixel où commence le remplissage
		A - couleur de motif à remplir (idem couleur de peinture de SCREEN 2)
	Sortie :	rien

Modifie : HL

Note : Il est nécessaire d'appeler SETATR (0011AH) puis PNTINI (00129H) avant d'utiliser cette routine.

PAINT fonctionne dans les SCREEN 2 à 8.

Un exemple d'utilisation :

```
10 CLEAR 100,&HD000
20 AD=&HD000
30 READ A$:IF a$= « FIN » THEN 100
40 POKE AD, VAL(« &H »+A$):AD=AD+1:GOTO 30
100 SCREEN 5:COLOR 2,1,1:CLS:LINE(&H5F,&H5F)-
(&H90,&H90),7,B:DEFUSR=&HD000:PRINT USR(0)
110 GOTO 110
1000 DATA 3E,09,CD,1A,01,01,80,00,11,80,00,3E,07,CD,29,
01,CD,E3,59,C9,FIN
```

Le programme en langage machine contenu à la ligne 1000 :

```
LD    A,9
CALL  SETATR
LD    BC, 080H
LD    DE, 080H
LD    A,7
CALL  PNTINI
CALL  PAINT
RET
```

3.3 LE BIOS EN SUB-ROM

Voici la suite du Bios à l'intérieur de la Sub-ROM. Si vous ne comprenez pas le sens de Sub-ROM, voyez le chapitre sur les slots pour plus d'explications sur la signification de la mémoire morte auxiliaire.

Mémoire Morte auxiliaire : (« Sub-ROM »)

00069H **PAINT** PAINT

Nom : PAINT

Adresse : 00069H/SUBROM

Type : MSX2

Rôle : remplissage d' une surface à l'écran

Entrée : HL - pointeur sur le texte Basic

Sortie : HL - pointeur réactualisé

Modifie : tout

Note : Valide dans les SCREEN 5 à 8

Exemple d'utilisation :

```
                ORG   0C000H
;
EXTROM          EQU   0015FH
PAINT           EQU   00069H
;
DEBUT:          LD    HL, DATA
                LD    IX, PAINT
                CALL  EXTROM
                RET
DATA:           DB    '(100,100),5,8'
                DB    00H
                END   DEBUT
```

0006DH **PSET** Pixel SET

Nom : PSET

Adresse : 0006DH/SUBROM

Type : MSX2

Rôle : allumage d'un pixel à l'écran

Entrée : HL - pointeur sur le texte Basic

Sortie : HL - pointeur réactualisé

Modifie : tout

Note : Valide dans les SCREEN 5 à 8

Voir l'exemple de PAINT pour le fonctionnement.

00071H **ATRSCN** ATRibute SCaN

Nom : ATRSCN

Adresse : 00071H/SUBROM

Type : MSX2

Rôle : recherche d'une couleur

Entrée : HL - pointeur sur le texte Basic

Sortie : HL - pointeur réactualisé

Modifie : tout

Note : Valide dans les SCREEN 5 à 8.

00075H **GLINE** Graphic LINE

Nom :	GLINE
Adresse :	00075H/SUBROM
Type :	MSX2
Rôle :	Tracé d'une ligne
Entrée :	HL - pointeur de texte Basic
Sortie :	HL - pointeur réactualisé
Modifie :	tout
Note :	Valide dans les SCREEN 5 à 8.
	Cette routine étant difficile à mettre en œuvre, il est préférable d'utiliser LINE (058FCH en main-ROM) ou DOGRAPH (00085H en Sub-ROM).

00079H **DOBOXF** DO BOX Fill

Nom :	DOBOXF
Adresse :	00079H/SUBROM
Type :	MSX2
Rôle :	tracé d'un rectangle plein
Entrée :	HL - pointeur de texte Basic
	BC - abscisse du pixel de départ
	DE - ordonnée du pixel de départ
	GXPOS - abscisse du pixel d'arrivée
	GYPOS - ordonnée du pixel d'arrivée
Sortie :	HL - pointeur réactualisé
Modifie :	tout
Note :	Valide dans les SCREEN 5 à 8.
	Cette routine étant difficile à mettre en œuvre, il est préférable d'utiliser BOXFIL (058C1H en main-ROM) ou NVBXFL (000CDH en Sub-ROM).

0007DH **DOLINE** DO LINE

Nom :	DOLINE
Adresse :	0007DH/SUBROM
Type :	MSX2
Rôle :	tracé de ligne
Entrée :	HL - pointeur de texte Basic
	BC - abscisse du pixel de départ
	DE - ordonnée du pixel de départ
	GXPOS - abscisse du pixel d'arrivée
	GYPOS - ordonnée du pixel d'arrivée
Sortie :	HL - pointeur réactualisé
Modifie :	tout
Note :	Valide dans les SCREEN 5 à 8.
	Cette routine étant difficile à mettre en œuvre, il est préférable d'utiliser LINE (058FCH en main-ROM) ou DOGRAPH (00085H en Sub-ROM).

00081H **BOXLIN**

Nom :	BOXLIN
Adresse :	00081H/SUBROM
Type :	MSX2
Rôle :	tracé d'un rectangle
Entrée :	HL - pointeur de texte Basic
	BC - abscisse du pixel de départ
	DE - ordonnée du pixel de départ
	GXPOS - abscisse du pixel d'arrivée
	GYPOS - ordonnée du pixel d'arrivée
Sortie :	HL - pointeur réactualisé
Modifie :	tout
Note :	Valide dans les SCREEN 5 à 8.
	Cette routine étant difficile à mettre en œuvre, il est préférable d'utiliser NVBXLN (000C9H en Sub-ROM).

00085H **DOGRPH** DO GRaPHic line

Nom :	DOGRPH
Adresse :	00085H/SUBROM
Type :	MSX2
Rôle :	tracé de ligne
Entrée :	BC - abscisse du pixel de départ
	DE - ordonnée du pixel de départ
	GXPOS - abscisse du pixel d'arrivée
	GYPOS - ordonnée du pixel d'arrivée
	ATRBYT - couleur à utiliser
	LOGOPR - opérateur logique à appliquer
Sortie :	rien
Modifie :	AF
Note :	Valide dans les SCREEN 5 à 8.

LOGOPR (« logical operation ») peut être :

0 - 0000 - IMP	8 - 1000 - TIMP
1 - 0001 - AND	9 - 1001 - TAND
2 - 0010 - OR	10 - 1010 - TAND
3 - 0011 - XOR	11 - 1011 - TXOR
4 - 0100 - NOT	12 - 1100 - TNOT

La seconde série d'opérateurs est identique à la première si ce n'est qu'un test est effectué pour savoir si le fond est en couleur 0. Si tel est le cas, l'opérateur n'agit pas.

Exemple d'utilisation :

```
            ORG   0C000H
;
EXTROM      EQU   0015FH
DOGRPH      EQU   00085H
GXPOS       EQU   0FCB3H
GYPOS       EQU   0FCB5H
ATRBYT      EQU   0F3F2H
LOGOPR      EQU   0FB02H
;
DEBUT:      LD    BC, 10H
            LD    DE, 50H
            LD    HL, 90H
            LD    (GXPOS), HL
            LD    (GYPOS), HL
            LD    A,7
            LD    (ATRBYT), A
            XOR   A
            LD    (LOGOPR), A
            LD    IX, DOGRPH
```

```
        CALL  EXTROM
        RET
        END   DEBUT
```

00089H **GRPPRT** GRaPhic PRinT

Nom :	GRPPRT
Adresse :	00089H/SUBROM
Type :	MSX2
Rôle :	affichage d'un caractère sur écran graphique
Entrée :	A - code ASCII du caractère
	GRPACX - abscisse du pixel où doit s'afficher le caractère
	GRPACY - ordonnée du pixel où doit s'afficher le caractère
	ATRBYT - couleur du caractère
	LOGOPR - opérateur logique
Sortie :	rien
Modifie :	rien
Note :	Valide dans les SCREEN 5 à 8.
	Voir DOGRPH pour l'explication de LOGOPR.

0008DH **SCALXY** SCALe X&Y

Nom :	SCALXY
Adresse :	0008DH/SUBROM
Type :	MSX2
Rôle :	vérification et éventuellement ajustement des valeurs du curseur graphique
Entrée :	BC - valeur de X (0-0FFFFH)
	DE - valeur de Y (0-0FFFFH)
Sortie :	BC - valeur de X valide
	DE - valeur de Y valide
	indicateur carry à 1 - une des coordonnées était hors de l'écran
Modifie :	AF
Note :	Lorsque l'une des deux coordonnées est négative, cette routine prend 0 comme nouvelle coordonnée. De même, si une coordonnée dépasse la valeur maximale autorisée (511 ou 255 pour X, 191 ou 211 pour Y), la routine prend la valeur maximale comme nouvelle coordonnée.

00091H **MAPXYC** MAP X&Y to Current

Nom : MAPXYC
Adresse : 00091H/SUBROM
Type : MSX2
Rôle : transformation des coordonnées d'un pixel
Entrée : BC - abscisse du pixel à allumer
DE - ordonnée du pixel à allumer
Sortie : en SCREEN 3 :
HL et CLOC - adresse en mémoire vidéo du pixel
A et CMASK - masque à appliquer à l'octet ci-dessus pour allumer le pixel qui nous intéresse
dans les SCREEN 5 à 8 :
HL et CLOC - abscisse du pixel à allumer
A et CMASL - ordonnée du pixel à allumer
Modifie : F
Note : Valide dans les SCREEN 3 et 5 à 8.

00095H **READC** READ Current

Nom : READC
Adresse : 00095H/SUBROM
Type : MSX2
Rôle : lecture de l'attribut du pixel actuel
Entrée : CLOC - abscisse du pixel actuel
CMASK - ordonnée du pixel actuel
Sortie : A - attribut
Modifie : AF
Note : Valide dans les SCREEN 3 et 5 à 8.

00099H **SETATR** SET ATRibute

Nom : SETATR
Adresse : 00099H/SUBROM
Type : MSX2
Rôle : modification de l'octet d'attribut (ATRBYT)
Entrée : A - valeur du nouvel attribut
Sortie : indicateur carry à 1 - attribut non valide (>15 sauf en SCREEN 8)
Modifie : F

0009DH **SETC** SET Current

Nom : SETC

Adresse : 0009DH/SUBROM

Type : MSX2

Rôle : allumage du pixel actuel

Entrée : CLOC - abscisse du pixel actuel

CMASK - ordonnée du pixel actuel

ATRBYT - couleur du pixel

Sortie : rien

Modifie : AF

Note : Valide dans les SCREEN 3 et 5 à 8.

000A1H **TRIGHTC** Test RIGHT Current

Nom : TRIGHTC

Adresse : 000A1H/SUBROM

Type : MSX2

Rôle : déplacement d'un pixel vers la droite après vérification de la validité des coordonnées

Entrée : CLOC et CMASK - coordonnées du pixel actuel

Sortie : CLOC et CMASK - nouvelles coordonnées

Modifie : AF

Note : Valide en SCREEN 3 uniquement.

000A5H **RIGHTC** RIGHT Current

Nom : RIGHTC

Adresse : 000A5H/SUBROM

Type : MSX2

Rôle : déplacement d'un pixel vers la droite

Entrée : CLOC et CMASK - coordonnées du pixel actuel

Sortie : CLOC et CMASK - nouvelles coordonnées

Modifie : AF

Note : Valide en SCREEN 3 uniquement.

000A9H **TLEFTC** Test LEFT Current

Nom : TLEFTC

Adresse : 000A9H/SUBROM

Type : MSX2

Rôle : déplacement d'un pixel vers la gauche après vérification de la validité des coordonnées

Entrée : CLOC et CMASK - coordonnées du pixel actuel

Sortie : CLOC et CMASK - nouvelles coordonnées

Modifie : AF

Note : Valide en SCREEN 3 et 5 à 8.

000ADH **LEFTC** LEFT Current

Nom : LEFTC

Adresse : 000ADH/SUBROM

Type : MSX2

Rôle : déplacement d'un pixel vers la gauche

Entrée : CLOC et CMASK - coordonnées du pixel actuel

Sortie : CLOC et CMASK - nouvelles coordonnées

Modifie : AF

Note : Valide en SCREEN 3 uniquement.

000B1H **TDOWNC** Test DOWN Current

Nom : TDOWNC

Adresse : 000B1H/SUBROM

Type : MSX2

Rôle : déplacement d'un pixel vers le bas après vérification de la validité des coordonnées

Entrée : CLOC et CMASK - coordonnées du pixel actuel

Sortie : CLOC et CMASK - nouvelles coordonnées

Modifie : AF

Note : Valide en SCREEN 3 et 5 à 8.

000B5H **DOWNC** DOWN Current

Nom :	DOWNC
Adresse :	000B5H/SUBROM
Type :	MSX2
Rôle :	déplacement d'un pixel vers le bas
Entrée :	CLOC et CMASK - coordonnées du pixel actuel
Sortie :	CLOC et CMASK - nouvelles coordonnées
Modifie :	AF
Note :	Valide en SCREEN 3 uniquement.

000B9H **TUPC** Test UP Current

Nom :	TUPC
Adresse :	000B9H/SUBROM
Type :	MSX2
Rôle :	déplacement d'un pixel vers le haut après vérification de la validité des coordonnées
Entrée :	CLOC et CMASK - coordonnées du pixel actuel
Sortie :	CLOC et CMASK - nouvelles coordonnées
Modifie :	AF
Note :	Valide en SCREEN 3 et 5 à 8.

000BDH **UPC** UP Current

Nom :	UPC
Adresse :	000BDH/SUBROM
Type :	MSX2
Rôle :	déplacement d'un pixel vers le haut
Entrée :	CLOC et CMASK - coordonnées du pixel actuel
Sortie :	CLOC et CMASK - nouvelles coordonnées
Modifie :	AF
Note :	Valide en SCREEN 3 uniquement.

000C1H **SCANR** SCAN Right

Nom :	SCANR
Adresse :	000C1H/SUBROM
Type :	MSX2
Rôle :	recherche vers la droite du premier pixel d'une autre couleur
Entrée :	DE - nombre de pixels sur lesquels la recherche doit être effectuée
Sortie :	DE - position du pixel recherché
Modifie :	tout
Note :	Voir SCANR en main-ROM pour plus d'explications.

000C5H **SCANL** SCAN Left

Nom :	SCANL
Adresse :	000C5H/SUBROM
Type :	MSX2
Rôle :	recherche vers la gauche du premier pixel d'une autre couleur
Entrée :	DE - nombre de pixels sur lesquels la recherche doit être effectuée
Sortie :	DE - position du pixel recherché
Modifie :	tout
Note :	Voir SCANL en main-ROM pour plus d'explications.

000C9H **NVBXLN**

Nom :	NVBXLN
Adresse :	000C9H/SUBROM
Type :	MSX2
Rôle :	tracé d'un rectangle à l'écran
Entrée :	BC - abscisse du pixel de départ
	DE - ordonnée du pixel de départ
	GXPOS - abscisse du pixel d'arrivée
	GYPOS - ordonnée du pixel d'arrivée
	ATRBYT - couleur à utiliser
	LOGOPR - opérateur logique à appliquer
Sortie :	rien
Modifie :	AF
Note :	Valide dans les SCRENN 5 à 8.
	Voir DOGRPH pour l'explication de LOGOPR

000CDH **NVBXFL**

Nom :	NVBXFL
Adresse :	000CDH/SUBROM
Type :	MSX2
Rôle :	tracé d'un rectangle plein à l'écran
Entrée :	BC - abscisse du pixel de départ
	DE - ordonnée du pixel de départ
	GXPOS - abscisse du pixel d'arrivée
	GYPOS - ordonnée du pixel d'arrivée
	ATRBYT - couleur à utiliser
	LOGOPR - opérateur logique à appliquer
Sortie :	rien
Modifie :	AF
Note :	Valide dans les SCRENN 5 à 8.
	Voir DOGRPH pour l'explication de LOGOPR

000D1H **CHGMOD** CHanGe MODe

Nom :	CHGMOD
Adresse :	000D1H/SUBROM
Type :	MSX2
Rôle :	changement de mode d'écran (SCREEN X)
Entrée :	A - SCREEN désiré (0-8)
Sortie :	rien
Modifie :	tout
Note :	La palette n'est pas initialisée par cette routine, utilisez la routine CHGMDP (001B5H en Sub-ROM) si vous avez besoin de l'initialisation de la palette.

000D5H **INITXT** INItialize TeXT mode

Nom :	INITXT
Adresse :	000D5H/SUBROM
Type :	MSX2
Rôle :	initialisation du mode texte 40x24 (SCREEN 0)
Entrée :	TXTNAM - adresse de la table des noms
	TXTCGP - adresse de la table des formes
Sortie :	rien
Modifie :	tout
Note :	Il n'est pas obligatoire de définit TXTNAM et TXTCGP car le MSX initialise leur valeur à l'allumage.

000D9H **INIT32** INItialize Text 32 mode

Nom :	INIT32
Adresse :	000D9H/SUBROM
Type :	MSX2
Rôle :	initialisation du mode texte 32x24 (SCREEN 1)
Entrée :	T32NAM - table des noms
	T32CGP - table des formes
	T32COL - table des couleurs
	T32ATR - table des attributs de sprites
	T32PAT - table des formes de sprites
Sortie :	rien
Modifie :	tout
Note :	Il n'est pas obligatoire de définit T32NAM, etc car le MSX initialise leur valeur à l'allumage.

000DDH **INIGRP** INItialize GRaPhic mode

Nom :	INIGRP
Adresse :	000DDH/SUBROM
Type :	MSX2
Rôle :	initialisation du mode graphique 256x192 (SCREEN 2)
Entrée :	GRPNAM - table des noms
	GRPCGP - table des formes
	GRPCOL - table des couleurs
	GRPATR - table des attributs de sprites
	GRPPAT - table des formes de sprites
Sortie :	rien
Modifie :	tout
Note :	Il n'est pas obligatoire de définit GRPNAM, etc car le MSX initialise leur valeur à l'allumage.

000E1H **INIMLT** INItialize MuLTicolor mode

Nom :	INIMLT
Adresse :	000E1H/SUBROM
Type :	MSX2
Rôle :	initialisation du mode graphique « multicolore » (SCREEN 3)
Entrée :	MLTNAM - table des noms
	MLTCGP - table des formes
	MLTCOL - table des couleurs
	MLTATR - table des attributs de sprites
	MLTPAT - table des formes de sprites
Sortie :	rien
Modifie :	tout
Note :	Il n'est pas obligatoire de définit MLTNAM, etc car le MSX initialise leur valeur à l'allumage.

000E5H **SETTXT** SET TeXT mode

Nom :	SETTXT
Adresse :	000E5H/SUBROM
Type :	MSX2
Rôle :	passage direct en mode texte 40x24
Entrée :	TXTNAM - adresse de la table des noms
	TXTCGP - adresse de la table des formes
Sortie :	rien
Modifie :	tout
Note :	Il n'est pas obligatoire de définit TXTNAM et TXTCGP car le MSX initialise leur valeur à l'allumage.

000E9H **SETT32** SET Text 32 mode

Nom :	SETT32
Adresse :	000E9H/SUBROM
Type :	MSX2
Rôle :	passage direct en mode texte 32x24
Entrée :	T32NAM - table des noms
	T32CGP - table des formes
	T32COL - table des couleurs
	T32ATR - table des attributs de sprites
	T32PAT - table des formes de sprites
Sortie :	rien
Modifie :	tout
Note :	Il n'est pas obligatoire de définit T32NAM, etc car le MSX initialise leur valeur à l'allumage.

000EDH **SETGRP** SET GRaPhic mode

Nom :	SETGRP
Adresse :	000EDH/SUBROM
Type :	MSX2
Rôle :	passage direct en mode graphique 256x192
Entrée :	GRPNAM - table des noms
	GRPCGP - table des formes
	GRPCOL - table des couleurs
	GRPATR - table des attributs de sprites
	GRPPAT - table des formes de sprites
Sortie :	rien
Modifie :	tout
Note :	Il n'est pas obligatoire de définit GRPNAM, etc car le MSX initialise leur valeur à l'allumage.

000F1H **SETMLT** SET MuLTicolor mode

Nom :	SETMLT
Adresse :	000F1H/SUBROM
Type :	MSX2
Rôle :	passage direct en mode graphique « multicolore »
Entrée :	MLTNAM - table des noms
	MLTCGP - table des formes
	MLTCOL - table des couleurs
	MLTATR - table des attributs de sprites
	MLTPAT - table des formes de sprites
Sortie :	rien
Modifie :	tout
Note :	Il n'est pas obligatoire de définit MLTNAM, etc car le MSX initialise leur valeur à l'allumage.

000F5H **CLRSPR** CLeaR SPRites

Nom :	CLRSPR
Adresse :	000F5H/SUBROM
Type :	MSX2
Rôle :	initialisation des sprites
Entrée :	SCRMOD - SCREEN actuel
Sortie :	rien
Modifie :	tout
Note :	La table des formes de sprites est effacée, les numéros de sprites sont mis aux numéros de plans, la couleur des sprites est mise à celle de la couleur du texte (FORCLR) et la position des sprites est réglée à 217.

000F9H **CALPAT** CALculate address of PATtern table

Nom :	CALPAT
Adresse :	000F9H/SUBROM
Type :	MSX2
Rôle :	calcul de l'adresse du début des formes d'un sprite
Entrée :	A - numéro de plan du sprite
Sortie :	HL - adresse cherchée
Modifie :	AF, DE, HL
Note :	Idem CALPAT en main-ROM.

000FDH **CALATR** CALculate address of ATRibute table

Nom :	CALATR
Adresse :	000FDH/SUBROM
Type :	MSX2
Rôle :	calcul de l'adresse du début des attributs d'un sprite
Entrée :	A - numéro de plan du sprite
Sortie :	HL - adresse cherchée
Modifie :	AF, DE, HL
Note :	Idem CALATR en main-ROM.

00101H **GSPSIZ** Get SPrite SIZE

Nom :	GSPSIZ
Adresse :	00101H/SUBROM
Type :	MSX2
Rôle :	calcul de la taille des sprites
Entrée :	rien
Sortie :	A - nombre d'octets pour un sprite (8 ou 32)
Modifie :	AF
Note :	Idem GSPSIZ en main-ROM.

00105H **GETPAT** GET PATtern

Nom :	GETPAT
Adresse :	00105H/SUBROM
Type :	MSX2
Rôle :	recherche du pattern d'un caractère
Entrée :	A - code ASCII du caractère
Sortie :	PATWRK - les huit octets qui définissent le pattern du caractère
Modifie :	tout

00109H **WRTVRM** WRiTe VRaM

Nom :	WRTVRM
Adresse :	00109H/SUBROM
Type :	MSX2
Rôle :	écriture dans une case mémoire de la VRAM
Entrée :	HL - adresse de case mémoire (0-0FFFFH)
	A - donnée à écrire
Sortie :	rien
Modifie :	AF
Note :	Cette routine permet l'accès à tout la VRAM, au contraire du WRTVRM de la main-ROM.

0010DH **RDVRM** ReaD VRaM

Nom :	RDVRM
Adresse :	0010DH/SUBROM
Type :	MSX2
Rôle :	lecture d'une case mémoire de la VRAM
Entrée :	HL - adresse de case mémoire (0-0FFFFH)
Sortie :	A - contenu de la case mémoire lue
Modifie :	AF
Note :	Cette routine permet l'accès à tout la VRAM, au contraire du RDVRM de la main-ROM.

00111H **CHGCLR** CHanGe CoLoR

Nom :	CHGCLR
Adresse :	00111H/SUBROM
Type :	MSX2
Rôle :	changement des couleurs à l'écran
Entrée :	FORCLR - couleur de texte
	BAKCLR - couleur de fond
	BDRCLR - couleur de marge
Sortie :	rien
Modifie :	tout
Note :	Idem CHGCLR de la main-ROM

00115H **CLS** CLear Screen

Nom :	CLS
Adresse :	00115H/SUBROM
Type :	MSX2
Rôle :	effacement de l'écran
Entrée :	rien
Sortie :	rien
Modifie :	tout

00119H **CLRTXT** CLeaR TeXT screen

Nom :	CLRTXT
Adresse :	00119H/SUBROM
Type :	MSX2
Rôle :	effacement de l'écran en mode texte
Entrée :	rien
Sortie :	rien
Modifie :	tout

0011DH **DSPFNK** DiSPlay FuNction Keys

Nom :	DSPFNK
Adresse :	0011DH/SUBROM
Type :	MSX2
Rôle :	affichage des touches de fonction
Entrée :	rien
Sortie :	rien
Modifie :	tout
Note :	Valide en SCREEN 0 et 1 uniquement.

00121H **DELLN0** DELete LiNe mode 0

Nom :	DELLN0
Adresse :	00121H/SUBROM
Type :	MSX2
Rôle :	effacement d'une ligne en mode texte
Entrée :	L - numéro de la ligne
Sortie :	rien
Modifie :	tout
Note :	Valide en SCREEN 0 uniquement.

00125H **INSLN0** INSert LiNe mode 0

Nom :	INSLN0
Adresse :	00125H/SUBROM
Type :	MSX2
Rôle :	Insertion d'une ligne en mode texte
Entrée :	L - numéro de la ligne
Sortie :	rien
Modifie :	tout
Note :	Valide en SCREEN 0 uniquement.

00129H **PUTVRM** PUT character in VRaM

Nom :	PUTVRM
Adresse :	00129H/SUBROM
Type :	MSX2
Rôle :	affichage d'un caractère en mode texte
Entrée :	C - code ASCII du caractère à afficher
	H - numéro de colonne
	L - numéro de ligne
Sortie :	rien
Modifie :	AF

0012DH **WRTVDP** WRiTe VDP

Nom :	WRTVDP
Adresse :	0012DH/SUBROM
Type :	MSX2
Rôle :	écriture dans un registre du VDP
Entrée :	C - numéro de registre (0-23 et 32-46)
	B - donnée à écrire
Sortie :	rien
Modifie :	AF, BC

00131H **VDPSTA** VDP STAtus

Nom :	VDPSTA
Adresse :	00131H/SUBROM
Type :	MSX2
Rôle :	lecture d'un des registres de statut du VDP
Entrée :	A - numéro de registre (0-9)
Sortie :	A - contenu du registre lu
Modifie :	F

00135H **KYKLOK** KeY Kana LOcK

Nom :	KYKLOK
Adresse :	00135H/SUBROM
Type :	MSX2
Rôle :	traitement du voyant et des touches KANA.
Entrée :	rien
Sortie :	rien
Modifie :	AF
Note :	Sans grand intérêt pour les MSX disponibles en France.

00139H **PUTCHR** PUT kana CHaRacter

Nom :	PUTCHR
Adresse :	00139H/SUBROM
Type :	MSX2
Rôle :	détection de l'appui sur une touche du clavier et transformation en code KANA.
Entrée :	indicateur zéro à 1 - pas de conversion
Sortie :	rien
Modifie :	tout
Note :	Sans grand intérêt pour les MSX disponibles en France.

0013DH **SETPAG** SET PAGe

Nom : SETPAG

Adresse : 0013DH/SUBROM

Type : MSX2

Rôle : changement de page graphique

Entrée : ACPAGE - page affichée à l'écran

DPPAGE - page sur laquelle on travaille

Sortie : rien

Modifie : AF

00141H **INIPLT** INItialize PaLeTte

Nom : INIPLT

Adresse : 00141H/SUBROM

Type : MSX2

Rôle : initialisation de la palette de couleurs

Entrée : rien

Sortie : rien

Modifie : AF, BC, DE

Note : équivalent de l'instruction Basic « COLOR=NEW »

00145H **RSTPLT** ReSTore PaLeTte

Nom : RSTPLT

Adresse : 00145H/SUBROM

Type : MSX2

Rôle : récupération de la palette depuis la VRAM

Entrée : rien

Sortie : rien

Modifie : AF, BC, DE

Note : équivalent de l'instruction Basic « COLOR=RESTORE »

00149H **GETPLT** GET PaLeTte

Nom : GETPLT

Adresse : 00149H/SUBROM

Type : MSX2

Rôle : récupération d'une couleur depuis la palette

Entrée : A - numéro de la couleur

Sortie : B - 4 bits de poids fort pour le rouge

4 bits de poids faible pour le bleu

C - 4 bits de poids faible pour le vert

Modifie : AF, DE

0014DH **SETPLT** SET PaLeTte

Nom : SETPLT

Adresse : 0014DH/SUBROM

Type : MSX2

Rôle : modification d'une couleur dans la palette

Entrée : D - numéro de la couleur (0-15)

A - 4 bits de poids fort pour le rouge

4 bits de poids faible pour le bleu

E - 4 bits de poids faible pour le vert

Sortie : rien

Modifie : AF

00151H **PUTSPR** PUT SPRite

Nom : PUTSPR

Adresse : 00151H/SUBROM

Type : MSX2

Rôle : affichage d'un sprite

Entrée : HL - pointeur de texte Basic

Sortie : HL - pointeur réactualisé

Modifie : tout

00155H **COLOR** COLOR

Nom : COLOR

Adresse : 00155H/SUBROM

Type : MSX2

Rôle : changement de couleurs, de palette, de couleur de sprites

Entrée : HL - pointeur de texte Basic

Sortie : HL - pointeur réactualisé

Modifie : tout

00159H **SCREEN** SCREEN

Nom : SCREEN

Adresse : 00159H/SUBROM

Type : MSX2

Rôle : modification de tous les paramètres définis par l'instruction Basic « SCREEN »

Entrée : HL - pointeur de texte Basic

Sortie : HL - pointeur réactualisé

Modifie : tout

0015DH **WIDTHS** WIDTH Screen

Nom : WIDTHS

Adresse : 0015DH/SUBROM

Type : MSX2

Rôle : modification de la largeur de l'écran

Entrée : HL - pointeur de texte Basic

Sortie : HL - pointeur réactualisé

Modifie : tout

00161H **VDP** VDP

Nom : VDP

Adresse : 00161H/SUBROM

Type : MSX2

Rôle : écriture dans un registre du VDP

Entrée : HL - pointeur de texte Basic

Sortie : HL - pointeur réactualisé

Modifie : tout

00165H **VDPF** VDP Fetch

Nom : VDPF
Adresse : 00165H/SUBROM
Type : MSX2
Rôle : lecture d'un registre du VDP
Entrée : HL - pointeur de texte Basic
Sortie : HL - pointeur réactualisé
Modifie : tout

00169H **BASE** BASE

Nom : BASE
Adresse : 00169H/SUBROM
Type : MSX2
Rôle : écriture dans un registre « base » du VDP
Entrée : HL - pointeur de texte Basic
Sortie : HL - pointeur réactualisé
Modifie : tout

0016DH **BASEF** BASE Fetch

Nom : BASEF
Adresse : 0016DH/SUBROM
Type : MSX2
Rôle : lecture d'un registre « base » du VDP
Entrée : HL - pointeur de texte Basic
Sortie : HL - pointeur réactualisé
Modifie : tout

00171H **VPOKE** Video POKE

Nom : VPOKE
Adresse : 00171H/SUBROM
Type : MSX2
Rôle : écriture dans la mémoire vidéo
Entrée : HL - pointeur de texte Basic
Sortie : HL - pointeur réactualisé
Modifie : tout

00175H **VPEEK** Video PEEK

Nom : VPEEK
Adresse : 00175H/SUBROM
Type : MSX2
Rôle : lecture de la mémoire vidéo
Entrée : HL - pointeur de texte Basic
Sortie : HL - pointeur réactualisé
Modifie : tout

00179H **SETS** SET Screen

Nom : SETS
Adresse : 00179H/SUBROM
Type : MSX2
Rôle : accès à tous les « SET » (SCREEN, ADJUST, TIME, etc)
Entrée : HL - pointeur de texte Basic
Sortie : HL - pointeur réactualisé
Modifie : tout

0017DH **BEEP** BEEP

Nom : BEEP
Adresse : 0017DH/SUBROM
Type : MSX2
Rôle : émission d'un bref bip sonore
Entrée : rien
Sortie : rien
Modifie : tout

00181H **PROMPT** PROMPT

Nom : PROMPT
Adresse : 00181H/SUBROM
Type : MSX2
Rôle : affichage de OK ou du message défini par l'utilisateur
Entrée : rien
Sortie : rien
Modifie : tout

00185H **SDFSCR**

Nom : SDFSCR

Adresse : 00185H/SUBROM

Type : MSX2

Rôle : récupération des options (largeur d'écran, couleurs, etc) sauvegardés dans la RAM CMOS de l'horloge

Entrée : rien

Sortie : rien

Modifie : tout

00189H **SETSCR** SET SCReen

Nom : SETSCR

Adresse : 00189H/SUBROM

Type : MSX2

Rôle : idem SDFSCR avec en plus l'affichage du message de départ à l'allumage

Entrée : rien

Sortie : rien

Modifie : tout

0018DH **SCOPY** Screen COPY

Nom : SCOPY

Adresse : 0018DH/SUBROM

Type : MSX2

Rôle : exécute tous les « COPY »

Entrée : HL - pointeur de texte Basic

Sortie : HL - pointeur réactualisé

Modifie : tout

00191H **BLTVV** BLock Transfer Vram to Vram

Nom : BLTVV

Adresse : 00191H/SUBROM

Type : MSX2

Rôle : transfert d'un bloc d'un endroit de mémoire vidéo à un autre

Entrée : HL - 0F562H

Sortie : rien

Modifie : tout

Note : Exemple d'utilisation :

COPY (32,16)-(128,96) TO (192,128)

```
                ORG   0C000H
;
EXTROM          EQU   0015FH
BLTVV           EQU   00191H
SX              EQU   0F562H        ;abscisse pixel de départ
SY              EQU   SX+2          ; ordonnée pixel de départ
DX              EQU   SY+2          ; abscisse pixel de destination
DY              EQU   DX+2          ; ordonnée pixel de destination
NX              EQU   DY+2          ; largeur du bloc à copier
NY              EQU   NX+2          ; hauteur du bloc à copier
ARG             EQU   NY+3
LOGOP           EQU   ARG+1         ; opérateur logique
;
DEBUT:          LD    HL, 20H
                LD    (SX), HL
                LD    HL, 10H
                LD    (SY), HL
                LD    HL, 0C0H
                LD    (DX), HL
                LD    HL, 80H
                LD    (DY), HL
                LD    HL, 80H-20H+1       ; calcul de la largeur
                LD    (NX), HL
                LD    HL, 60H-10H+1       ; calcul de la hauteur
                LD    (NY), HL
                XOR   A             ; ARG doit toujours être mis à zéro !
                LD    (ARG), A
                LD    A,3
                LD    (LOGOP),A     ; idem LOGOPR
;                                   ; ne pas oublier ceci
                LD    HL, SX
                LD    IX, BLTVV
                CALL  EXTROM
                RET
                END   DEBUT
```

L'ordinateur travaille comme s'il n'y avait qu'un écran géant dans lequel on déplace une fenêtre suivant le schéma :

```
        SCREEN 5             VRAM           SCREEN 6
************************ 00000H ************************
*(0,0)           (255,0)*        *(0,0)           (511,0)*
*        page 0         *        *        page 0         *
*(0,255)       (255,255)*        *(0,255)       (511,255)*
************************ 08000H ************************
*(0,256)       (255,256)*        *(0,256)       (511,256)*
*        page 1         *        *        page 1         *
*(0,511)       (255,511)*        *(0,511)       (511,511)*
************************ 10000H ************************
*(0,512)       (255,512)*        *(0,512)       (511,512)*
*        page 2         *        *        page 2         *
*(0,767)       (255,767)*        *(0,767 )      (511,767)*
************************ 18000H ************************
*(0,768)       (255,768)*        *(0,768)       (511,768)*
*        page 3         *        *        page 3         *
*(0,1023)     (255,1023)*        *(0,1023)     (511,1023)*
************************ 20000H ************************

        SCREEN 7             VRAM           SCREEN 8
************************ 00000H ************************
*(0,0)           (511,0)*        *(0,0)           (255,0)*
*        page 0         *        *        page 0         *
*(0,255)       (511,255)*        *(0,255)       (255,255)*
************************ 10000H ************************
*(0,256)       (511,256)*        *(0,256)       (255,256)*
*        page 1         *        *        page 1         *
*(0,511)       (511,511)*        *(0,511)       (255,511)*
************************ 20000H ************************
```

00195H **BLTVM** BLock Transfert Vram from Memory

Nom : BLTVM

Adresse : 00195H/SUBROM

Type : MSX2

Rôle : transfert de la mémoire centrale vers la mémoire vidéo

Entrée : HL - 0F562H

Sortie : rien

Modifie : tout

Note : Exemple d'utilisation :

COPY &HD000 TO (16,32)

```
            ORG   0C000H
;
EXTROM      EQU   0015FH
BLTVM       EQU   00195H
DPTR        EQU   0F562H        ; adresse en mémoire centrale
DX          EQU   DPTR+4        ; abscisse pixel de destination
DY          EQU   DX+2          ; ordonnée pixel de destination
ARG         EQU   DY+7
LOGOP       EQU   ARG+1         ; opérateur logique
;
DEBUT:      LD    HL, 0D000H
            LD    (DPTR), HL
            LD    HL, 10H
            LD    (DX), HL
            LD    HL, 20H
            LD    (DY), HL
            XOR   A             ; ARG doit toujours être mis à zéro !!!
            LD    (ARG), A
            LD    A,2
            LD    (LOGOP), A    ; Idem LOGOP
;                               ; ne pas oublier ceci
            LD    HL, DPTR
            LD    IX, BLTVM
            CALL  EXTROM
            RET
;
            ORG   0D000H
P. HORZ     DW    030H          ; nombre de pixels horizontaux
P. VERT     DW    00BH          ; nombre de pixels verticaux
NBOCT       DS    210H          ; nombre d'octets nécessaires
            END   DEBUT
```

Pour calculer NBOCT :

en SCREEN 5 - (NBOCT) = (P. HORZ)/2 * (P.VERT) +1
en SCREEN 6 - (NBOCT) = (P. HORZ)/4 * (P.VERT) +1
en SCREEN 7 - (NBOCT) = (P. HORZ)/2 * (P.VERT) +1
en SCREEN 8 - (NBOCT) = (P. HORZ) * (P.VERT)
Pour plus de précisions, voir la routine BLTVV.

00199H **BLTMV** BLock Transfert Memory from Vram

Nom : BLTMV
Adresse : 00199H/SUBROM
Type : MSX2
Rôle : transfert de la mémoire vidéo vers la mémoire centrale
Entrée : HL - 0F562H
Sortie : rien
Modifie : tout
Note : Exemple d'utilisation :

COPY (16,32) TO &HD000

```
        ORG   0C000H
;
EXTROM  EQU   0015FH
BLTMV   EQU   00199H
SX      EQU   0F562H      ; abscisse du pixel de départ
SY      EQU   SX+2        ; ordonnée du pixel de départ
DPTR    EQU   SY+2        ; adresse en mémoire centrale
NX      EQU   SY+6        ; abscisse du pixel de destination
NY      EQU   NX+2        ; ordonnée du pixel de destination
ARG     EQU   NY+3
LOGOP   EQU   ARG+1       ; opérateur logique
;
DEBUT:  LD    HL, 0D000H
        LD    (DPTR), HL
        LD    HL, 10H
        LD    (SX), HL
        LD    HL, 20H
        LD    (SY), HL
        LD    HL, 60H-10H+1      ; calcul de la largeur
        LD    (NX), HL
        LD    HL, 40H-20H+1      ; calcul de la hauteur
        LD    (NY), HL
        XOR   A           ; ARG doit toujours être mis à zéro !!!
        LD    (ARG), A
        LD    A,2
```

```
                LD      (LOGOP), A   ; Idem LOGOP
;                                    ; ne pas oublier ceci
                LD      HL, SX
                LD      IX, BLTMV
                CALL    EXTROM
                RET
                END     DEBUT
```

Vous trouverez en 0D000H et 0D001H la largeur du bloc, en 0D002H et 0D003H la hauteur du bloc. Pour calculer le nombre d'octets utilisés à partir de 0D004H pour stocker le bloc, appliquez la formule adéquate :

en SCREEN 5 - nb octets = largeur/2 * hauteur +1

en SCREEN 6 - nb octets = largeur/4 * hauteur +1

en SCREEN 7 - nb octets = largeur/2 * hauteur +1

en SCREEN 8 - nb octets = largeur * hauteur

Pour plus de précisions, voir la routine BLTVV.

0019DH **BLTVD** BLock Transfert Vram from Disk

Nom : BLTVD

Adresse : 0019DH/SUBROM

Type : MSX2

Rôle : transfert de la disquette vers la mémoire vidéo

Entrée : HL - 0F562H

Sortie : rien

Modifie : tout

Note : Exemple d'utilisation :

COPY « A: ESSAI.SC7 » to (16,32)

```
                ORG     0C000H
;
EXTROM          EQU     0015FH
BLTVD           EQU     0019DH
FNPTR           EQU     0F562H       ; pointeur sur le nom de fichier
DX              EQU     FNPTR+4      ; abscisse du pixel de destination
DY              EQU     DX+2         ; ordonnée du pixel de destination
ARG             EQU     DY+7
LOGOP           EQU     ARG+1        ; opérateur logique
;
DEBUT:          LD      HL, NOM
                LD      (FNPTR), HL
                LD      HL, 10H
                LD      (DX), HL
                LD      HL, 20H
```

```
                LD      (DY), HL
                XOR     A               ; ARG doit toujours être mis à zéro !!!
                LD      (ARG), A
                LD      (LOGOP), A      ; Idem LOGOP
;                                       ; ne pas oublier ceci
                LD      HL, FNPTR
                LD      IX, BLTVD
                CALL    EXTROM
                RET
;
NOM:            DB      022H,'A:ESSAI.SC7',022H,000H
                END     DEBUT
```

Si vous appelez cette routine depuis le Basic, faites-le par un DEFUSR=&HC000 : A$=USR(0). N'oubliez pas le dollar. Dans A$, vous aurez le nom du fichier.

Pour plus de précisions, voir la routine BLTVV.

001A1H **BLTDV** BLock Transfert Disk from Vram

Nom : BLTDV

Adresse : 001A1H/SUBROM

Type : MSX2

Rôle : transfert de la mémoire vidéo vers la disquette

Entrée : HL - 0F562H

Sortie : rien

Modifie : tout

Note : Exemple d'utilisation :

COPY (16,32)-(96,128) TO « A:ESSAI.SC7 »

```
        ORG     0C000H

;
EXTROM          EQU     0015FH
BLTDV           EQU     001A1H
SX              EQU     0F562H          ;  abscisse du pixel de départ
SY              EQU     SX+2            ; ordonnée du pixel de départ
FNPTR           EQU     SY+2            ; pointeur sur le nom de fichier
NX              EQU     SY+6    ; abscisse du pixel de destination
NY              EQU     NX+2            ; ordonnée du pixel de destination
ARG             EQU     NY+3
;
DEBUT:          LD      HL, NOM
                LD      (FNPTR), HL
                LD      HL, 10H
                LD      (SX), HL
                LD      HL, 20H
                LD      (SY), HL
```

```
                LD      HL, 60H-10H+1       ; calcul de la largeur
                LD      (NX), HL
                LD      HL, 80H-20H+1       ; calcul de la hauteur
                LD      (NY), HL
                XOR     A               ; ARG doit toujours être mis à zéro !!!
                LD      (ARG), A
;                                       ; ne pas oublier ceci
                LD      HL, SX
                LD      IX, BLTDV
                CALL    EXTROM
                RET
;
NOM:            DB      022H,'A:ESSAI.SC7',022H,000H
                END     DEBUT
```

Si vous appelez cette routine depuis le Basic, faites-le par un DEFUSR=&HC000 : A$=USR(0). N'oubliez pas le dollar. Dans A$, vous aurez le nom du fichier.

Pour plus de précisions, voir la routine BLTVV.

001A5H **BLTMD** BLock Transfert Memory from Disk

Nom : BLTMD

Adresse : 001A5H/SUBROM

Type : MSX2

Rôle : transfert de la disquette vers la mémoire centrale

Entrée : HL - 0F562H

Sortie : rien

Modifie : tout

Note : Exemple d'utilisation :

```
COPY « A:ESSAI.SC7 » TO <0D000H>
```

```
        ORG     0C000H
;
EXTROM          EQU     0015FH
BLTMD           EQU     001A5H
FNPTR           EQU     0F562H      ; pointeur sur le nom de fichier
SPTR            EQU     FNPTR+4     ; adresse de départ en mémoire
EPTR            EQU     FNPTR+6     ; adresse de fin en mémoire
;
DEBUT:          LD      HL, NOM
                LD      (FNPTR), HL
                LD      HL, 0D000H
                LD      (SPTR), HL
                LD      HL, 0D020H
                LD      (EPTR), HL
;                                   ; ne pas oublier ceci
```

```
                LD      HL, FNPTR
                LD      IX, BLTMD
                CALL    EXTROM
                RET
;
NOM:            DB      022H,'A:ESSAI.SC7',022H,000H
                END     DEBUT
```

Si vous appelez cette routine depuis le Basic, faites-le par un DEFUSR=&HC000 : A$=USR(0). N'oubliez pas le dollar. Dans A$, vous aurez le nom du fichier.

001A9H **BLTDM** BLock Transfert Disk from Memory

Nom : BLTDM

Adresse : 001A9H/SUBROM

Type : MSX2

Rôle : transfert de la mémoire centrale vers la disquette

Entrée : HL - 0F562H

Sortie : rien

Modifie : tout

Note : Exemple d'utilisation :

COPY <0D000H> TO « A:ESSAI.SC7 »

```
        ORG     0C000H
;
EXTROM          EQU     0015FH
BLTDM           EQU     001A9H
SPTR            EQU     0F562H          ; adresse de départ en mémoire
EPTR            EQU     SPTR+2          ; adresse de fin en mémoire
FNPTR           EQU     SPTR+4          ; pointeur sur le nom de fichier
;
DEBUT:          LD      HL, NOM
                LD      (FNPTR), HL
                LD      HL, 0D000H
                LD      (SPTR), HL
                LD      HL, 0D020H
                LD      (EPTR), HL
;                                       ; ne pas oublier ceci
                LD      HL, SPTR
                LD      IX, BLTDM
                CALL    EXTROM
                RET
;
NOM:            DB      022H,'A:ESSAI.SC7',022H,000H
                END     DEBUT
```

Si vous appelez cette routine depuis le Basic, faites-le par un

DEFUSR=&HC000 : A$=USR(0). N'oubliez pas le dollar. Dans A$, vous aurez le nom du fichier.

001ADH **NEWPAD** NEW get touch PAD

Nom : NEWPAD

Adresse : 001ADH/SUBROM

Type : MSX2

Rôle : lecture de la tablette graphique, du stylo optique, de la souris et du track-ball (cat)

Entrée : A - numéro de l'opération à effectuer (0-19)

Sortie : A - résultat de l'opération

Modifie : tout

Note : Sur MSX2, il est possible d'appeler indifféremment cette routine ou GTPAD en main-ROM.

Voici la liste des opérations possibles :

0 - tablette graphique dans le port joystick n°1 prête (0FFH si oui)

1 - abscisse lue sur la tablette graphique dans le port joystick n°1

2 - ordonnée lue sur la tablette graphique dans le port joystick n°1

3 - le crayon est-il sur la tablette dans le port joystick n°1 (0FFH si oui, 0 si non)

4 - ablette graphique dans le port joystick n°2 prête (0FFH si oui)

5 - abscisse lue sur la tablette graphique dans le port joystick n°2

6 - ordonnée lue sur la tablette graphique dans le port joystick n°2

7 - le crayon est-il sur la tablette dans le port joystick n°2 (0FFH si oui, 0 si non)

8 - le crayon optique est-il prêt à être lu (0FFH si oui)

9 - abscisse du crayon optique

10 - ordonnée du crayon optique

11 - le bouton du crayon optique est-il enfoncé (0FFH si oui, 0 si non)

12 - initialisation de la souris ou du track-ball dans le port joystick n°1 (renvoie toujours 0FFH)

13 - déplacement horizontal de la souris ou du track-ball dans le port joystick n°1

14 - déplacement vertical de la souris ou du track-ball dans le port joystick n°1

15 - rien (renvoie toujours 0)

16 - initialisation de la souris ou du track-ball dans le port joystick n°2 (renvoie toujours 0FFH)

17 - déplacement horizontal de la souris ou du track-ball dans le port joystick n°2

18 - déplacement vertical de la souris ou du track-ball dans le port joystick n°2

19 - rien (renvoie toujours 0)

Le Bios reconnaît automatiquement si c'est une souris ou un track-ball qui est connecté.

Il est indispensable de faire un appel à la fonction 12 (ou 16 pour le port joystick n°2) avant de lire le déplacement de la souris ou du track-ball. Dans le cas contraire, vous obtiendrez des valeurs qui n'ont aucun sens.

001B1H **GETPUT** GET time/date & PUT kanji

Nom : GETPUT

Adresse : 001B1H/SUBROM

Type : MSX2

Rôle : récupération de l'heure et de la date et traitement des caractères kanji

Entrée : HL - pointeur de texte Basic

Sortie : rien

Modifie : tout

001B5H **CHGMDP** CHanGe MoDe vdP

Nom : CHGMDP

Adresse : 001B5H/SUBROM

Type : MSX2

Rôle : modifications des registres du VDP pour changer de SCREEN

Entrée : A - numéro du SCREEN (0-8)

Sortie : rien

Modifie : tout

Note : Par rapport à CHGMOD, cette routine initialise la palette.

001B9H **RESV1** RESerVed 1

Nom : RESV1

Adresse : 001B9H/SUBROM

Type : MSX2

Rôle : Réservé aux versions futures de MSX

Entrée : //

Sortie : //

Modifie : //

001BDH **KNJPRT** KaNJi PRinT

Nom : KNJPRT

Adresse : 001BDH/SUBROM

Type : MSX2

Rôle : affichage d'un caractère kanji à l'écran

Entrée : BC - code du caractère kanji

A - type d'affichage

Sortie : rien

Modifie : AF

001F5H **REDCLK** REaD CLocK

Nom : REDCLK

Adresse : 001F5H/SUBROM

Type : MSX2

Rôle : lecture de la mémoire vive non-volatile contenue dans l'horloge interne

Entrée : C - adresse mémoire

Sortie : A - contenu de la case mémoire lue (quatre bits de poids faible)

Modifie : F

Note : L'adresse mémoire se décompose ainsi :

```
b7      b6      b5      b4      b3      b2      b1      b0
!       !       !       !       !       !       !       !
!       !       !       !       +-------+-------+-------+-- adresse (0-12)
!       !       +-------+---------------------------------- mode (2 ou 3)
+-------+-------------------------------------------------- inutilisés (=0)
```

En mode 2 :

adresse	quatre bits
00	identificateur (1010 si la RAM est initialisée)
01	SET ADJUST X (-8 à 7)
02	SET ADJUST Y (-8 à 7)
03	SCREEN (0 ou 1)
04	WIDTH - quatre bits de poids faible
05	WIDTH - trois bits de poids fort, le bit 3 reste libre car 0 à 127 suffit pour coder la largeur d'écran
06	couleur de texte (0-15)
07	couleur de fond (0-15)
08	couleur de la marge (0-15)
09	quatre flags : bit 0 : key - 0 off / 1 on bit 1 : key click - 0 off / 1 on bit2 : imprimante - 0 MSX / 1 non MSX

	bit3 : baud - 0 = 1200 / 1 = 2400
10	SET BEEP : bits 0 et 1 - volume (0-3) bits 2 et 3 - type de BEEP (0-3)
11	couleur de la page de présentation (0-3) les bits 2 et 3 sont disponibles pour l'utilisateur
12	code de pays (0-15) : 0 - Japon / 1 - USA / 2 - International / 3 - GB / 4 - France / 5 - Allemagne / 6 - Italie / 7 - Espagne / 8 - Emirats arabes / 9 - Corée / 10 - URSS / 11 - nd / 12 - nd / 13 - nd / 14 - nd / 15 - nd (nd : non défini au 05/02/1986)

En mode 3 :

adresse	quatre bits
00	identificateur (0-15) : 0 - TITLE / 1 - PASSWORD / 2 - PROMPT 3 à 15 - non définis au 05/02/1986
01	quatre bits de poids faible de l'octet 1
01	quatre bits de poids fort de l'octet 1
01	quatre bits de poids faible de l'octet 2
01	quatre bits de poids fort de l'octet 2
01	quatre bits de poids faible de l'octet 3
01	quatre bits de poids fort de l'octet 3
01	quatre bits de poids faible de l'octet 4
01	quatre bits de poids fort de l'octet 4
01	quatre bits de poids faible de l'octet 5
01	quatre bits de poids fort de l'octet 5
01	quatre bits de poids faible de l'octet 6
01	quatre bits de poids fort de l'octet 6

Exemple d'utilisation :

```
        ORG   0C000H
;
EXTROM  EQU   0015FH
RDCLK   EQU   001F5H
;
DEBUT:  LD    C,02CH        ; mode 2 - adresse 12
        LD    IX, RDCLK
        CALL  EXTROM
        RET                 ; lors du retour A contient 4
```

001F9H	**WRTCLK**	WRiTe CLocK
	Nom :	WRTCLK
	Adresse :	001F9H/SUBROM
	Type :	MSX2
	Rôle :	écriture dans la mémoire vive non-volatile de l'horloge interne
	Entrée :	C - adresse mémoire
		A - donnée à écrire (quatre bits)
	Sortie :	rien
	Modifie :	F
	Note :	Voir RDCLK (ci-dessus) pour le détail du fonctionnement de cette routine.

4 LES VARIABLES SYSTEME ET LES HOOKS

4.1 INTRODUCTION AUX VARIABLES SYSTEME

L'utilité des variables système n'est plus à démontrer. La plupart des trucs et astuces sont basés sur une bonne connaissance de celles-ci. Il devient ainsi facile de combler les quelques lacunes du Basic. Quant au programmeur en langage machine, il peut la plupart du temps ignorer complètement les variables système mais il ne faut pas négliger que la mémoire vive qui leur est réservée peut parfois constituer une zone de travail appréciable avec les très gros programmes et les inévitables problèmes de place mémoire qui les accompagnent. Enfin, le programmeur qui souhaite développer des routines en langage machine qui coexistent avec l'interpréteur Basic devra être particulièrement attentif à cette zone mémoire.

4.2 LA LISTE DES VARIABLES SYSTEME

Adresse	Nom	Long.	Fonction
0F39AH	USRTAB	20	Liste des adresses définies par « DEFUSRx= »
0F3AEH	LINL40	1	Longueur d'une ligne en mode 0, 37 par défaut
0F3AFH	LINL32	1	Longueur d'une ligne en mode 1, 29 par défaut
0F3B0H	LINLEN	1	Longueur de la ligne actuelle (modifié par « WIDTH »)
0F3B1H	CRTCNT	1	Longueur de la page actuelle
0F3B2H	CLMLST	1	Utilisé par TAB
0F3B3H	TXTNAM	2	Adresse de la table des noms en mode 0
0F3B5H	TXTCOL	2	Inutilisée
0F3B7H	TXTCGP	2	Adresse de la table des formes en mode 0
0F3B9H	TXTATR	2	Inutilisée
0F3BBH	TXTPAT	2	Inutilisée

0F3BDH	T32NAM	2	Adresse de la table des noms en mode 1
0F3BFH	T32COL	2	Adresse de la table des couleurs en mode 1
0F3C1H	T32CGP	2	Adresse de la table des formes en mode 1
0F3C3H	T32ATR	2	Adresse de la table des attributs de sprites en mode 1
0F3C5H	T32PAT	2	Adresse de la table des formes de sprites en mode 1
0F3C7H	GRPNAM	2	Adresse de la table des noms en mode 2
0F3C9H	GRPCOL	2	Adresse de la table des couleurs en mode 2
0F3CBH	GRPCGP	2	Adresse de la table des formes en mode 2
0F3CDH	GRPATR	2	Adresse de la table des attributs de sprites en mode 2
0F3CFH	GRPPAT	2	Adresse de la table des formes de sprites en mode 2
0F3D1H	MLTNAM	2	Adresse de la table des noms en mode 3
0F3D3H	MLTCOL	2	Adresse de la table des couleurs en mode 3
0F3D5H	MLTCGP	2	Adresse de la table des formes en mode 3
0F3D7H	MLTATR	2	Adresse de la table des attributs de sprites en mode 3
0F3D9H	MLTPAT	2	Adresse de la table des formes de sprites en mode 3
0F3DBH	CLIKSW	1	« Click switch » : 0 pour enlever le clic des touches 1 pour mettre le clic des touches
0F3DCH	CSRY	1	Ordonnée du curseur
0F3DDH	CSRX	1	Abscisse du curseur
0F3DEH	CNSDFG	1	0 pour utiliser tout l'écran 1 pour afficher le contenu des touches de fonction
0F3DFH	RG0SAV	1	Contenu du registre 0 du VDP
0F3E0H	RG1SAV	1	Contenu du registre 1 du VDP
0F3E1H	RG2SAV	1	Contenu du registre 2 du VDP
0F3E2H	RG3SAV	1	Contenu du registre 3 du VDP
0F3E3H	RG4SAV	1	Contenu du registre 4 du VDP
0F3E4H	RG5SAV	1	Contenu du registre 5 du VDP
0F3E5H	RG6SAV	1	Contenu du registre 6 du VDP
0F3E6H	RG7SAV	1	Contenu du registre 7 du VDP
0F3E7H	STATFL	1	Contenu du registre de statut du VDP (registre 8)
0F3E8H	TRGFLG	1	« Trigger flag » : (11111111B) utilisé par la routine qui vérifie l'état des boutons des joysticks

0F3E9H	FORCLR	1	Couleur de texte, blanc par défaut
0F3EAH	BAKCLR	1	Couleur de fond, bleu par défaut
0F3EBH	BDRCLR	1	Couleur de la bordure, bleu par défaut
0F3ECH	MAXUPD	3	Saut, en 0 par défaut
0F3EFH	MINUPD	3	Saut, en 0 par défaut
0F3F2H	ATRBYT	1	Octet d'attribut, 15 par défaut
0F3F3H	QUEUES	2	Adresse de la table des « queue » QUEUTL
0F3F5H	FRCNEW	1	Mémoire de travail, 255 par défaut
0F3F6H	SCNCNT	1	Intervalle entre deux scrutations du clavier
0F3F7H	REPCNT	1	Intervalle avant de commencer la fonction de répétition automatique, 50 par défaut. Réglé en permanence par le Basic.
0F3F8H	PUTPNT	2	Adresse du premier emplacement libre dans le buffer
0F3FAH	GETPNT	2	Adresse de la prochaine donnée issue du buffer clavier
0F3FCH	CS120	10	Zone de travail pour la cassette
0F406H	LOW	2	Paramètre cassette. Largeur du 0
0F408H	HIGH	2	Paramètre cassette. Largeur du 1
0F40AH	HEADER	1	Paramètre cassette. Longueur « header » / 128
0F40BH	ASPCT1	2	Utilisé par « CIRCLE » pour la forme du cercle
0F40DH	ASPCT2	2	Idem que ASPCT1
0F40FH	ENDPRG	5	Fin de programme bidon pour « RESUME NEXT »

fin des variables système initialisées à l'allumage de l'ordinateur.

0F414H	ERRFLG	1	Numéro de la dernière erreur
0F415H	LPTPOS	1	Position de la tête de l'imprimante, 0 au départ
0F416H	PRTFLG	1	Flag de sortie sur imprimante. 0 pas de sortie
0F417H	NTMSXP	1	Flag de type d'imprimante : 0 imprimante MSX 1 autre imprimante
0F418H	RAWPRT	1	Flag de type caractère : 0 pour un caractère codé 1 pour un caractère non codé
0F419H	VLZADR	2	Adresse du caractère remplacé par l'instruction « VAL »

0F41BH	VLZDAT	1	Caractère remplacé par 0 avec « VAL »
0F41CH	CURLIN	2	Numéro de la ligne en cours d'exécution
0F41FH	KBUF	318	« Crunch buffer » utilisé par le Basic
0F55DH	BUFMIN	1	Petit buffer intermédiaire
0F55EH	BUF	256	Buffer du mode direct
0F660H	ENDBUF	1	Pour empêcher les lignes trop longues de dépasser
0F661H	TTYPOS	1	Position finale stockée ici
0F662H	DIMFLG	1	Flag pour savoir si un « DIM » est en cours
0F663H	VALTYP	1	Type de variable
0F664H	OPRTYP	1	Type d'opérateur
0F664H	DORES	1	Flag pour savoir si on peut comprimer des mots-clés (par exemple « DATA, READ, PRINT » ne peut être comprimé)
0F665H	DONUM	1	Flag utilisé pour la compression des nombres
0F666H	CONTXT	2	Sauvegarde temporaire du pointeur de texte
0F668H	CONSAV	1	Sauvegarde temporaire du code de l'instruction en cours
0F669H	CONTYP	1	Constante sauvée : type
0F66AH	CONLO	8	Constante sauvée : valeur
0F672H	MEMSIZ	2	Adresse de la dernière case mémoire disponible sous Basic. Modifié par l'instruction « CLEAR »
0F674H	STKTOP	2	Adresse de la dernière case mémoire utilisable par la pile. Modifié par « CLEAR »
0F676H	TXTTAB	2	Adresse de début des programmes Basic
0F678H	TEMPPT	2	Pointeur du Basic
0F67AH	TEMPST	30	Stockage provisoire
0F698H	DSCTMP	3	Adresse du premier octet libre dans la table des chaînes de caractères
0F69BH	FRETOP	2	Adresse de fin de table de chaînes de caractères
0F69DH	TEMP3	2	Mémoire de travail
0F69FH	TEMP8	2	Mémoire de travail
0F6A1H	ENDFOR	2	Sauvegarde du pointeur de texte après un « FOR » pour pouvoir revenir au moment du « NEXT »
0F6A3H	DATLIN	2	Numéro de ligne du « DATA » en cours
0F6A5H	SUBFLG	1	Flag pour « FOR » et les fonctions définies par l'utilisateur
0F6A6H	FLGINP	1	Flag pour « INPUT » et « READ »
0F6A7H	TEMP	2	Mémoire de travail pour « INPUT », « FOR-NEXT », « LET » et ^C
0F6A9H	PTRFLG	1	Flag de mode : 0 pas de numéro de ligne à transformer 1 numéro de ligne à transformer
0F6AAH	AUTFLG	1	Flag pour « AUTO » 0 pas de mode auto 1 mode auto en cours

0F6ABH	AUTLIN	2	Ligne actuelle en mode auto
0F6ADH	AUTINC	2	Valeur de l'incrémentation en mode auto
0F6AFH	SAVTXT	2	Sauvegarde du pointeur de texte après une erreur pour « RESUME »
0F6B1H	SAVSTK	2	Sauvegarde de l'adresse de la pile après une erreur afin d'être dans la bonne situation au moment du « RESUME »
0F6B3H	ERRLIN	2	Numéro de la ligne à laquelle la dernière erreur s'est produite
0F6B5H	DOT	2	Numéro de ligne actuelle. Utilisé dans « LIST »
0F6B7H	ERRTXT	2	Pointeur de texte utilisé par « RESUME »
0F6B9H	ONELIN	2	Numéro de ligne définie par « ON ERROR GOTO »
0F6BAH	ONEFLG	1	Flag d'erreur : 0 mode normal 1 traitement d'erreur en cours
0F6BCH	TEMP2	2	Mémoire de travail pour le programme d'évaluation de formules
0F6BEH	OLDLIN	2	Ancien numéro de ligne, après un ^C, un « STOP » ou un « END »
0F6C0H	OLDTXT	2	Ancien pointeur de texte. Le pointeur est dirigé sur l'instruction suivant celle où s'est produit l'arrêt
0F6C2H	VARTAB	2	Pointeur sur le début des variables simples
0F6C4H	ARYTAB	2	Pointeur sur le début des variables en tableaux (« arrays »)
0F6C6H	STREND	2	Adresse de la fin de la zone des variables
0F6C8H	DATPTR	2	Pointeur pour les « DATA ». Modifié par « RESTORE »
0F6CAH	DEFTBL	26	Type de variable (entier, chaîne, simple ou double précision) défini par « DEFSTR », « DEFINT », « DEFSNG », « DEFDBL » pour chacune des 26 lettres de l'alphabet

Zone de travail pour manipuler les paramètres de fonctions définies par l'utilisateur (« DEF FNX= », « PRINT FNX(p) » par exemple)

0F6E4H	PRMSTK	2	Nombre d'octets disponibles pour les définitions
0F6E6H	PRMLEN	2	Nombre d'octets actuellement utilisés
0F6E8H	PARM1	100	Définition des paramètres
0F74CH	PRMPRV	2	Pointeur pour le bloc de paramètres précédent
0F74EH	PRMLN2	2	Taille du bloc de paramètres en cours d'élaboration
0F750H	PARM2	100	Zone pour sauvegarder les blocs en cours de création
0F7B4H	PRMFLG	1	Flag pour savoir si PARM1 a été fouillé
0F7B5H	ARYTA2	2	Point d'arrêt pour une recherche simple
0F7B7H	NOFUNS	1	Flag de fonction : 0 si aucune fonction n'est active
0F7B8H	TEMP9	2	Mémoire de travail
0F7BAH	FUNACT	2	Nombre de fonctions actives
0F7BCH	SWPTMP	8	Mémoire de travail pour l'instruction « SWAP »
0F7C4H	TRCFLG	1	Flag de mode : 0 pas de « TRACE » en cours, 1 mode « TRACE » en exécution

0F7C5H	FBUFFR	43	Buffer utilisé par les routines mathématiques
0F7F0H	DECTMP	2	Mémoire de travail
0F7F2H	DECTM2	2	Mémoire de travail pour les divisions
0F7F4H	DECCNT	1	Mémoire de travail pour les divisions
0F7F6H	DAC	16	Accumulateur décimal
0F806H	HOLD8	48	Sauvegarde temporaire pendant les multiplications décimales
0F836H	HOLD2	8	Idem ci-dessus
0F83EH	HOLD	8	Idem ci-dessus
0F847H	ARG	16	Accumulateur décimal secondaire
0F857H	RNDX	8	Dernier nombre aléatoire généré
0F85FH	MAXFIL	1	Nombre maximum de fichiers autorisé. Modifié par « MAXFILES »
0F860H	FILTAB	2	Pointeur sur l'adresse des données du fichier
0F862H	NULBUF	2	Pointeur sur le buffer du fichier 0
0F864H	PTRFIL	2	Pointeur sur les sonnées du fichier sélectionné
0F866H	RUNFLG	1	Flag de « RUN » : 0 pour un chargement normal 1 pour une exécution automatique
0F866H	FILNAM	11	Nom de fichier pour « DIRSRC »
0F871H	FILNM2	11	Nom du second fichier lors d'un « NAME... AS ... »
0F87CH	NLONLY	1	Flag de chargement : 0 en mode normal 1 pendant un chargement
0F87DH	SAVEND	2	Fin d'une sauvegarde binaire
0F87FH	FNKSTR	160	Contenu des touches de fonction
0F91FH	CGPNT	3	Emplacement de la forme du caractère en ROM
0F922H	NAMBAS	2	Adresse de la table des noms actuelle
0F924H	CGPBAS	2	Adresse de la table des formes actuelle
0F926H	PATBAS	2	Adresse de la table des formes de sprites actuelle
0F928H	ATRBAS	2	Adresse de la table des attributs de sprites actuelle
0F92AH	CLOC	2	Adresse du pixel actuel
0F92CH	CMASK	1	Masque du pixel actuel
0F92DH	MINDEL	2	Mémoire de travail
0F92FH	MAXDEL	2	Mémoire de travail
0F931H	ASPECT	2	Aspect pour l'instruction « CIRCLE »
0F933H	CENCNT	2	Mémoire de travail pour « CIRCLE »
0F935H	CLINEF	1	Flag pour dessiner au centre de l'écran
0F936H	CNPNTS	2	Nombre de points à allumer
0F938H	CPLOTF	1	Flag de polarité
0F939H	CPCNT	2	1/8 du nombre de points dans le cercle
0F93BH	CPCNT8	2	Nombre de points dans le cercle

0F93DH	CRCSUM	2	Somme du cercle
0F93FH	CSTCNT	2	Total de départ
0F941H	CSCLXY	1	Mémoire de travail pour ?
0F942H	CSAVEA	2	Adresse du premier pixel de couleur différente. Utilisé par l'instruction « PAINT »
0F944H	CSAVEM	1	Masque du premier pixel de couleur différente. Utilisé par l'instruction « PAINT »
0F945H	CXOFF	2	Décalage horizontal depuis la position sauvegardée
0F947H	CYOFF	2	Décalage vertical depuis la position sauvegardée
0F949H	LOHMSK	1	Mémoire de travail pour « PAINT »
0F94AH	LOHDIR	1	Idem ci-dessus
0F94BH	LOHADR	2	Idem ci-dessus
0F94DH	LOHCNT	2	Idem ci-dessus
0F94FH	SKPCNT	2	Idem ci-dessus
0F951H	MOVCNT	2	Idem ci-dessus
0F953H	PDIREC	1	Direction dans laquelle on peint
0F954H	LFPROG	1	Mémoire de travail pour « PAINT »
0F955H	RTPROG	1	Idem ci-dessus
0F956H	MCLTAB	2	Mémoire de travail pour « MACLNG »
0F958H	MCLFLG	1	Flag indiquant un « PLAY » ou un « DRAW »
0F959H	QUETAB	24	Quatre queues de six octets chacune
0F971H	QUEBAK	4	Utilisé par « BCKQ »
0F975H	VOICAQ	128	Queue musicale pour la voie A
0F9F5H	VOICBQ	128	Queue musicale pour la voie B
0FA75H	VOICCQ	128	Queue musicale pour la voie C
0FAF5H	R2SIQ	64	Queue d'entrée du RS-232
0FAF5H	DPPAGE	1	Page affichée
0FAF6H	ACPAGE	1	Page active (sur laquelle on écrit)
0FAF7H	AVCSAV	1	Sauvegarde du port de contrôle AV
0FAF8H	EXBRSA	1	Slot dans lequel se trouve la Sub-ROM
0FAF9H	CHRCNT	1	Compteur de caractères pour une conversion Roma-kana
0FAFAH	ROMA	2	Sauvegarde de caractères lors des conversions Roma-kana
0FAFCH	MODE	1	Flag de conversion Roma-kana / taille de la VRAM
0FAFDH	-------	1	Non utilisé, réservé à des applications futures
0FAFEH	XSAVE	2	Sauvegarde de l'abscisse du curseur graphique lors de l'utilisation de la souris, du cat ou du stylo optique
0FAB0H	YSAVE	2	Sauvegarde de l'ordonnée du curseur graphique lors de l'utilisation de la souris, du cat ou du stylo optique

0FAB2H	LOGOPR	1	« LOGical OPeration code », mode de tracé : 0 pour IMP, 1 pour AND, 2 pour OR, 3 pour XOR, 4 pour NOT
0FAB3H	TOCNT	1	Donnée
0FAB4H	RSFCB	1	FCB (« File Control Block ») de la RS-232
0FAF5H	RSIQLN	1	Donnée
0FAF8H	SUBROM	1	Emplacement (slot) de la mémoire morte auxiliaire
0FB35H	PRSCNT	1	Mémoire de travail pour l'instruction « PLAY »
0FB36H	SAVSP	2	Sauvegarde de l'adresse de la pile pendant l'exécution de la musique
0FB38H	VOICEN	1	Numéro de la voie actuelle, utilisé par « PLAY »
0FB39H	SAVVOL	2	Sauvegarde du volume lors des pauses
0FB3BH	MCLLEN	1	Mémoire de travail pour « PLAY »
0FB3CH	MCLPTR	2	Idem ci-dessus
0FB3EH	QUEUEN	1	Numéro de la queue actuelle
0FB3FH	MUSICF	1	Flag d'interruption musicale
0FB40H	PLYCNT	1	Nombre de « PLAY » à exécuter simultanément
0FB41H	VCBA	37	Données pour la voie 0
0FB66H	VCBB	37	Données pour la voie 1
0FB8BH	VCBC	37	Données pour la voie 2
0FBB0H	ENSTOP	1	Démarrage à chaud possible si différent de 0
0FBB1H	BASROM	1	Programme Basic en ROM si différent de 0
0FBB2H	LINTTB	24	Table des lignes terminées : 0 pour une ligne en cours, une valeur non nulle pour une ligne terminée
0FBCAH	FSTPOS	2	Première position, utilisé par « QINLIN »
0FBCCH	COSAV	1	Sauvegarde du caractère lors de l'affichage du curseur
0FBCDH	FNKSWI	1	Indique quelle série de touches de fonction est affichée
0FBCEH	FNKFLG	10	Dix flags utilisés par l'instruction « ON KEY GOSUB... » pour dérouter ou non
0FBD8H	ONGSBF	1	Flag pour « ON... GOSUB »
0FBD9H	CLIKFL	1	Flag pour savoir si le click a déjà eu lieu
0FBDAH	OLDKEY	11	Statut de l'ancienne touche
0FBE5H	NEWKEY	11	Statut de la nouvelle touche
0FBF0H	KEYBUF	40	Buffer pour le codage d'une touche
0FC18H	LINWRK	40	Zone de travail pour la gestion de l'écran
0FC40H	PATWRK	8	Zone de travail pour le convertisseur noms -> formes
0FC48H	BOTTOM	2	Adresse du début de la RAM

0FC4AH	HIMEM	2	Adresse de la dernière case disponible en RAM
0FC4CH	TRPTBL	78	« Trap table »
0FC9AH	RTYCNT	1	Contrôle d'interruption
0FC9BH	INTFLG	1	Flag d'interruption
0FC9CH	PADX	1	Valeur en X du paddle (manette spéciale)
0FC9DH	PADY	1	Valeur en Y du paddle
0FC9EH	JIFFY	2	Petit compteur sympathique
0FCA0H	INTVAL	2	Valeur de l'intervalle lors d'une instruction « ON INTERVAL=... GOSUB... »
0FCA2H	INTCNT	2	Compteur utilisé par l'instruction « ON INTERVAL=... GOSUB... »
0FCA4H	LOWLIM	1	Utilisé durant la lecture cassette
0FCA5H	WINWID	1	Idem ci-dessus
0FCA6H	GRPHED	1	En-tête pour la sortie des caractères graphiques
0FCA7H	ESCCNT	1	Compteur pour une séquence d'escape
0FCA8H	INSFLG	1	Flag de mode insertion
0FCA9H	CSRSW	1	« Cursor display switch » : flag pour le curseur
0FCAAH	CSTYLE	1	Forme du curseur : 0 pour le curseur entier 1 pour un petit curseur (insert)
0FCABH	CAPST	1	Flag de mode : 0 pour être en minuscules 1 pour être en majuscules
0FCACH	KANAST	1	Flag de mode kana
0FCADH	KANAMD	1	Flag spécifique au mode kana
0FCAEH	FLBMEM	1	Flag de chargement : 0 si un programme Basic se charge
0FCAFH	SCRMOD	1	Mode d'écran actuel
0FCB0H	OLDSCR	1	Ancien mode d'écran. Utilisé pour savoir dans quel mode texte il faut repasser après un mode graphique
0FCB1H	CASPRV	1	Mémoire de travail pour la cassette
0FCB2H	BRDATR	1	Couleur de la frontière pour l'instruction « PAINT »
0FCB3H	GXPOS	2	Abscisse du curseur graphique
0FCB5H	GYPOS	2	Ordonnée du curseur graphique
0FCB7H	GRPACX	2	Accumulateur graphique X
0FCB9H	GRPACY	2	Accumulateur graphique Y
0FCBBH	DRWFLG	1	Flag pour l'instruction « DRAW »
0FCBCH	DRWSCL	1	Taille du zoom pour l'instruction « DRAW »
0FCBDH	DRWANG	1	Angle pour l'instruction « DRAW »
0FCBEH	RUNBNF	1	Flag d'I/O : 255 entrée/sortie binaire en cours
0FCBFH	SAVENT	2	Adresse de début spécifiée par l'instruction « BSAVE »
0FCC1H	MNROM	1	Emplacement (slot) de la mémoire morte principale

0FCC1H	EXPTBL	4	Un flag pour chacun des slots primaires : 255 si le slot est expandé en slots secondaires
0FCC5H	SLTTBL	4	Etat actuel pour chaque slot expandé
0FCC9H	SLTATR	64	Attributs de chaque slot
0FD09H	SLTWRK	128	Zone de travail pour chaque slot
0FD89H	PROCNM	16	Nom des instructions supplémentaires (pour « CALL »)
0FD99H	DEVICE	1	Numéro de « device » pour une cartouche (0-3)

4.3 QUELQUES EXEMPLES D'UTILISATION DES VARIABLES SYSTEME

- Sous Basic, vous désirez couper un bout d'écran afin de réserver quelques lignes (sur l'exemple des touches de fonction). Il vous suffit de faire :

  POKE &HF3B1, 24-nn   nn étant le nombre de lignes à réserver

- Il est parfois utile de connaître la largeur de l'écran (« WIDTH ») dans un programme d'application. Essayez :

  L=PEEK (&HF3B0)

- Dans le même ordre d'idées, le nombre maximum de fichiers ouvrables (défini par l'instruction « MAXFILES ») s'obtient par :

  M=PEEK(&HF85F)

- Qui n'a jamais connu le problème d'un buffer clavier qui renvoie des caractères au mauvais moment. Effacer ce buffer est souvent utile :

  En Basic :

  ```
  POKE &HF3FA, PEEK(&HF3F8)
  POKE &HF3FB,PEEK(&HF3F9)
  ```

  En langage machine :

  ```
  LD    HL, (PUTPNT)
  LD    (GETPNT),HL
  ```

- Voici un truc un peu plus sophistiqué : l'interdiction de l'action combinée des touches CRTL+STOP peut facilement être obtenue en faisant croire au MSX que votre programme se trouve en cartouche de mémoire morte.

  POKE &HFBB1,1 fera parfaitement l'affaire.

- Au contraire, pour ceux qui désirent accéder à des programmes Basic protégés contre le CTRL + STOP, il leur suffit de taper :

  POKE &HFBB0,1 avant de charger le programme.

  Puis, le programme ayant démarré, il faut enfoncer simultanément les touches CTRL, SHIFT, GRAPH et CODE. Le MSX rendra alors la main.

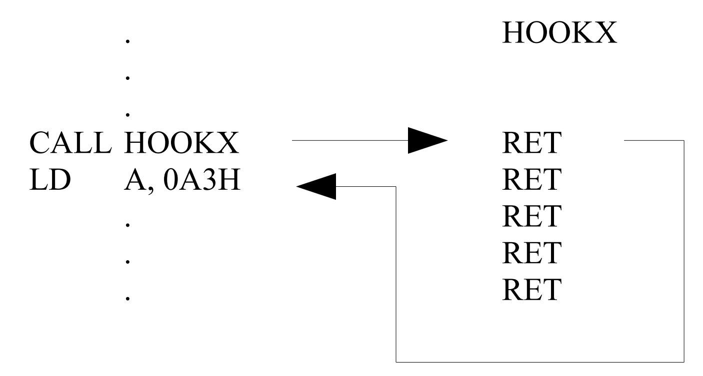
- Pour ceux que la présence du curseur rassure, ils peuvent essayer :
  POKE &HFCA9,1   en mode direct.
  Pour se rendre compte de l'effet obtenu, il suffit de rentrer le petit programme suivant :
  10 PRINT « Hello »
  20 GOTO 20   et de faire RUN.

- Le dernier exemple est un peu plus conséquent que les autres. Il s'agit d'utiliser la zone réservée à la définition des touches fonction au mieux de manière à pouvoir assigner n'importe quelle chaîne de caractères à une touche. Notons qu'on peut mettre au maximum 39 caractères par touche, sachant que l'on ne dispose de toutes manières que de 160 octets et que si l'on étend les contenu d'une touche de fonction, ce ne peut être qu'aux dépens de la suivante.
  Le petit programme Basic suivant illustre cet exemple :

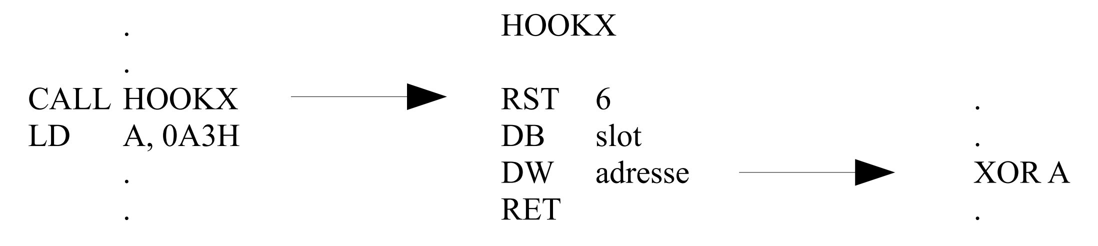
```
10 AD=&HF87F:CLS
20 READ A$: IF LEN(A$)>39 THEN PRINT « Impossible! »:END
30 FOR I=1 TO LEN (A$)
40 I$=MID$(A$, I, 1)
50 POKE AD-1+I,ASC(I$)
60 NEXT I
70 POKE AD-1+I,0:END
100 DATA « C'est un peu long jeune homme ... »
```

4.4 LA LISTE DES HOOKS

Un hook est une zone mémoire de cinq octets en mémoire vive (RAM). Il permet d'étendre les fonctions en mémoire morte. Au départ, les cinq octets contiennent tous 0C9H (RET), comme dans l'exemple suivant :



Cependant, il est facile de modifier les 5 octets afin de détourner la fonction en ROM, comme ceci :



Pour illustrer le fonctionnement des hooks, je vous propose un exemple en Basic qui utilise le hook H.LIST pour empêcher l'accès à la liste du programme.

```
10 POKE &HFF89, &HE1          On met dans H.LIST :
20 POKE &HFF8A, &HC3
30 POKE &HFF8B, &H9B                POP   HL
40 POKE &HFF8C, &H40                JP    0409BH
```

Il suffit de faire POKE &HFF89,201 pour restituer son pouvoir à la commande LIST.

Voici la liste des hooks pour les MSX1 et MSX2. Elle comprend le nom du hook, le nom de la routine qui l'appelle, ainsi que son utilité :

0FD9AH **H.KEYI**

Nom : H.KEYI

Appel : au début de la routine qui gère les interruptions

Fonct. : permet de gérer des interruptions supplémentaires (comme la RS-232)

0FD9FH **H.TIMI**

Nom : H.TIMI

Appel : routine qui gère les interruptions du timer

Fonct. : permet de gérer des interruptions supplémentaires grâce au timer

0FDA4H **H.CHPU**

Nom : H.CHPU

Appel : routine CHPUT (« Character outPUT »), sortie d'un caractère à l'écran

Fonct. : permet de sortir le caractère sur un périphérique autre que le moniteur vidéo

0FDA9H **H.DSPC**

Nom : H.DSPC

Appel : au début de la routine DSPCSR (« DiSPlay CurSoR »), affichage du curseur

Fonct. : permet l'accès à d'autres périphériques que le moniteur vidéo

0FDAEH **H.ERAC**

Nom : H.ERAC

Appel : routine ERACSR (« ERAse CurSoR »), effacement du curseur

Fonct. : permet l'accès à d'autres périphériques que le moniteur vidéo

0FDB3H **H.DSPF**

Nom : H.DSPF

Appel : routine DSPFNK (« DiSPlay FuNction Keys »), affichage du contenu des touches de fonction

Fonct. : permet l'accès à d'autres périphériques que le moniteur vidéo

0FDB8H **H.ERAF**

Nom : H.ERAF

Appel : routine ERAFNK (« ERAse FuNction Keys »), effacement du contenu des touches de fonction

Fonct. : permet l'accès à d'autres périphériques que le moniteur vidéo

0FDBDH **H.TOTE**

Nom : H.TOTE

Appel : routine TOTEXT (« force screen TO TEXT mode »), passage en mode texte (voir Bios)

Fonct. : permet l'accès à d'autres périphériques que le moniteur vidéo

0FDC2H **H.CHGE**

Nom : H.CHGE

Appel : routine CHGET (« CHaracter GET »), lecture d'un caractère au clavier

Fonct. : permet l'accès à d'autres périphériques d'entrée que le clavier

0FDC7H **H.INIP**

Nom : H.INIP

Appel : routine INIPAT (« INItialize PATtern »), remplissage de la table des formes en mode texte

Fonct. : permet de modifier le jeu de caractères lorsque l'on revient en mode texte (après SCREEN 2 par exemple)

0FDCCH **H.KEYC**

Nom : H.KEYC

Appel : routine KEYCOD (« KEY CODer »), lecture clavier

Fonct. : permet d'intercepter la lecture du clavier. Lorsque le hook H.KEYC est appelé, l'accumulateur contient dix fois le numéro de ligne plus le numéro de colonne de la touche enfoncée dans la matrice clavier

0FDD1H **H.KYEA**

Nom : H.KYEA

Appel : routine KYEASY (« KeY EASY »), conversion d'un caractère lu au clavier

Fonct. : permet de modifier la manière dont une touche est interprétée sous Basic

0FDD6H **H.NMI**

Nom : H.NMI

Appel : routine NMI (« Non Maskable Interrupt »), interruption non masquable

Fonct. : permet le traitement des interruptions non masquables

0FDDBH **H.PINL**

Nom :	H.PINL
Appel :	routine PINLIN (« Program INput LINe »), entrée au clavier d'une ligne de programme
Fonct. :	permet d'utiliser les 80 colonnes, ou un autre périphérique d'entrée que le clavier

0FDE0H **H.QLIN**

Nom :	H.QLIN
Appel :	routine QINLIN (« Question mark and INput LINe »), affichage d'un point d'interrogation puis entrée au clavier
Fonct. :	permet d'utiliser les 80 colonnes, ou un autre périphérique d'entrée que le clavier

0FDE5H **H.INLI**

Nom :	H.INLI
Appel :	routine INLIN (« INput LINe »), entrée au clavier
Fonct. :	permet d'utiliser les 80 colonnes, ou un autre périphérique d'entrée que le clavier

0FDEAH **H.ONGO**

Nom :	H.ONGO
Appel :	routine ONGOTP (« ONGOto Procedure »), pour les instructions Basic de type ON GOTO, ON GOSUB
Fonct. :	permet de détourner ce type d'instructions

0FDEFH **H.DSKO**

Nom :	H.DSKO
Appel :	routine DSKO$ (« DiSK Output »), utilisée par l'instruction DSKO$ du Disk-Basic
Fonct. :	détourné pour obtenir un accès au lecteur de disquettes

0FDF4H **H.SETS**

Nom :	H.SETS
Appel :	routine SETS (« SET attributeS »)
Fonct. :	détourné pour obtenir un accès au lecteur de disquettes

0FDF9H **H.NAME**

Nom : H.NAME

Appel : routine NAME (« reNAME »), utilisée par l'instruction NAME...AS du Disk-Basic

Fonct. : détourné pour obtenir un accès au lecteur de disquettes

0FDFEH **H.KILL**

Nom : H.KILL

Appel : routine KILL (« KILL file »), utilisée par l'instruction KILL du Disk-Basic

Fonct. : détourné pour obtenir un accès au lecteur de disquettes

0FE03H **H.IPL**

Nom : H.IPL

Appel : routine IPL (« Initial Program Load »)

Fonct. : détourné pour obtenir un accès au lecteur de disquettes

0FE08H **H.COPY**

Nom : H.COPY

Appel : routine COPY (« COPY files »), utilisée par l'instruction COPY du Disk-Basic

Fonct. : détourné pour obtenir un accès au lecteur de disquettes

0FE0DH **H.CMD**

Nom : H.CMD

Appel : routine CMD (« CoMmanD »), routine non utilisée sur MSX1 et MSX2

Fonct. : aucune sur MSX1 et MSX2; hook libre, disponible pour le programmeur

0FE12H **H.DSKF**

Nom : H.DSKF

Appel : routine DSKF (« DiSK Free »), utilisée par l'instruction DSKF$ du Disk-Basic

Fonct. : détourné pour obtenir un accès au lecteur de disquettes

0FE17H **H.DSKI**

Nom : H.DSKI

Appel : routine DSKI « DiSK Input »), utilisée par l'instruction DSKI$ du Disk-Basic

Fonct. : détourné pour obtenir un accès au lecteur de disquettes

0FE1CH **H.ATTR**

Nom : H.ATTR

Appel : Routine ATTR$ (« ATTRibute »)

Fonct. : détourné pour obtenir un accès au lecteur de disquettes

0FE21H **H.LSET**

Nom : H.LSET

Appel : routine LSET (« Left SET »), utilisée par l'instruction LSET du Disk-Basic

Fonct. : détourné pour obtenir un accès au lecteur de disquettes

0FE26H **H.RSET**

Nom : H.RSET

Appel : routine RSET (« Right SET »), utilisée par l'instruction RSET du Disk-Basic

Fonct. : détourné pour obtenir un accès au lecteur de disquettes

0FE2BH **H.FIEL**

Nom : H.FIEL

Appel : routine FIELD, utilisée par l'instruction FIELD du Disk-Basic

Fonct. : détourné pour obtenir un accès au lecteur de disquettes

0FE30H **H.MKI$**

Nom : H.MKI$

Appel : routine MKI$ (« MaKe Integer »), utilisée par l'instruction MKI$ du Disk-Basic

Fonct. : détourné pour obtenir un accès au lecteur de disquettes

0FE35H **H.MKS$**

Nom : H.MKS$

Appel : routine MKS$ (« MaKe Simple »), utilisée par l'instruction MKS$ du Disk-Basic

Fonct. : détourné pour obtenir un accès au lecteur de disquettes

0FE3AH **H.MKD$**

Nom : H.MKD$

Appel : routine MKD$ (« MaKe Double »), utilisée par l'instruction MKD$ du Disk-Basic

Fonct. : détourné pour obtenir un accès au lecteur de disquettes

0FE3FH **H.CVI**

Nom : H.CVI

Appel : routine CVI (« ConVert Integer »), utilisée par l'instruction CVI du Disk-Basic

Fonct. : détourné pour obtenir un accès au lecteur de disquettes

0FE44H **H.CVS**

Nom : H.CVS

Appel : routine CVS (« ConVert Single »), utilisée par l'instruction CVS du Disk-Basic

Fonct. : détourné pour obtenir un accès au lecteur de disquettes

0FE49H **H.CVD**

Nom : H.CVD

Appel : routine CVD (« ConVert Double »), utilisée par l'instruction CVD du Disk-Basic

Fonct. : détourné pour obtenir un accès au lecteur de disquettes

0FE4EH **H.GETP**

Nom : H.GETP

Appel : routine GETPTR « GET file PoinTeR »), positionnement sur un fichier

Fonct. : utilisé par le système d'exploitation des disquettes

0FE53H **H.SETF**

Nom : H.SETF

Appel : routine SETFIL (« SET FILe »), positionnement d'un pointeur sur un fichier ouvert au préalable

Fonct. : utilisé par le système d'exploitation des disquettes

0FE58H **H.NOFO**

Nom : H.NOFO

Appel : routine NOFOR (« NO FOR clause »), utilisée par l'instruction OPEN du Basic

Fonct. : utilisé par le système d'exploitation des disquettes

0FE5DH **H.NULO**

Nom : H.NULO

Appel : routine NULOPN (« NULl file OPen »), utilisée par les instructions LOAD, KILL, MERGE, etc

Fonct. : utilisé par le système d'exploitation des disquettes

0FE62H **H.NTFL**

Nom : H.NTFL

Appel : routine NTFL0 (« NoT FiLe number 0 »), utilisée par l'instruction CLOSE du Basic

Fonct. : utilisé par le système d'exploitation des disquettes

0FE67H **H.MERG**

Nom : H.MERG

Appel : routine MERGE (« MERGE program files »), utilisée par l'instruction MERGE du Basic

Fonct. : utilisé par le système d'exploitation des disquettes

0FE6CH **H.SAVE**

Nom : H.SAVE

Appel : routine SAVE, utilisée au début de l'instruction SAVE du Basic

Fonct. : utilisé par le système d'exploitation des disquettes

0FE71H **H.BINS**

Nom : H.BINS

Appel : routine BINSAV (« BINary SAVe »), sauvegarde d'une zone mémoire en binaire

Fonct. : utilisé par le système d'exploitation des disquettes

0FE76H **H.BINL**

Nom : H.BINL

Appel : routine BINLOD (« BINary LOaD »), chargement d'une zone mémoire en binaire

Fonct. : utilisé par le système d'exploitation des disquettes

0FE7BH **H.FILE**

Nom : H.FILE

Appel : routine FILES, utilisée par l'instruction FILES du Disk-Basic

Fonct. : utilisé par le système d'exploitation des disquettes

0FE80H **H.DGET**

Nom : H.DGET

Appel : routine DGET (« Disk GET »), utilisée par l'instruction GET du Disk-Basic

Fonct. : utilisé par le système d'exploitation des disquettes

0FE85H **H.FILO**

Nom : H.FILO

Appel : routine FILOU1 (« FILe OUt 1 »), sortie dans un fichier

Fonct. : utilisé par le système d'exploitation des disquettes

0FE8AH **H.INDS**

Nom : H.INDS

Appel : routine INDSKC (« INput DiSK Character »)

Fonct. : utilisé par le système d'exploitation des disquettes

0FE8FH **H.RSLF**

Nom : H.RSLF

Appel : routine qui sélectionne l'ancien lecteur de disquettes comme lecteur actuel

Fonct. : utilisé par le système d'exploitation des disquettes

0FE94H **H.SAVD**

Nom : H.SAVD

Appel : routine sauvegarde le lecteur actuel, utilisée par les instructions LOF, LOC, EOF, FPOS, etc du Disk-Basic

Fonct. : utilisé par le système d'exploitation des disquettes

0FE99H **H.LOC**

Nom : H.LOC

Appel : routine LOC (« LOCation »), utilisée lors de l'instruction LOC du Disk-Basic

Fonct. : utilisé par le système d'exploitation des disquettes

0FE9EH **H.LOF**

Nom : H.LOF

Appel : routine LOF (« Length Of File »), utilisée lors de l'instruction LOF du Disk-Basic

Fonct. : utilisé par le système d'exploitation des disquettes

0FEA3H **H.EOF**

Nom : H.EOF

Appel : routine EOF (« End Of File), utilisée lors de l'instruction EOF du Disk-Basic

Fonct. : utilisé par le système d'exploitation des disquettes

0FEA8H **H.FPOS**

Nom : H.FPOS

Appel : routine FPOS (« File POSition »)

Fonct. : utilisé par le système d'exploitation des disquettes

0FEADH **H.BAKU**

Nom : H.BAKU

Appel : routine BAKUPT (« BAKe UP »)

Fonct. : utilisé par le système d'exploitation des disquettes

0FEB2H **H.PARD**

Nom : H.PARD

Appel : routine PARDEV (« PARse DEVice name »)

Fonct. : permet de changer le nom du device « logical device name »

0FEB7H **H.NODE**

Nom : H.NODE

Appel : routine NODEVN (« NO DEVice Name »), pas de nom trouvé dans la table des DEVICE

Fonct. : permet de traiter le cas où le device n'est pas trouvé

0FEBCH **H.POSD**

Nom : H.POSD

Appel : routine POSDSK (« POSsible DiSK »)

Fonct. : utilisé par le système d'exploitation des disquettes

0FEC1H **H.DEVN**

Nom : H.DEVN

Appel : routine DEVNAM (« DEVice NAMe »)

Fonct. : aucune, hook non utilisé

0FEC6H **H.GEND**

Nom : H.GEND

Appel : routine GENDSP (« GENeral device DISPatcher »), traitement d'un ce non lié au disque

Fonct. : permet d'ajouter de nouveaux devices

0FECBH **H.RUNC**

Nom : H.RUNC

Appel : routine RUNC (« RUN Clear »), utilisée lors des instructions NEW et du Basic

Fonct. : détournement des instructions NEW et RUN

0FED0H **H.CLEA**

Nom : H.CLEA

Appel : routine CLEARC (« CLEAR Clear »), utilisée lors de la remise à zéro des variables Basic

Fonct. : permet d'éviter l'effacement des variables Basic

0FED5H **H.LOPD**

Nom : H.LOPD

Appel : routine LOPDFT (« LOoP and set DeFaulT »), initialisation de la table des variables

Fonct. : permet de détourner la routine

0FEDAH **H.STKE**

Nom : H.STKE

Appel : routine STKERR (« STacK ERRor »), utilisée lors des instructions Basic FOR et GOSUB

Fonct. : lors de l'initialisation du Disk-Basic ou du MSX-DOS, le MSX vérifie si ce hook a été modifié par une éventuelle cartouche utilisant le lecteur de disquette (contenu de 0FEDAH différent de 0C9H). Si tel est le cas, le MSX saute au hook (JP 0FEDAH) lorsque l'installation du DOS est terminée

0FEDFH **H.ISFL**

Nom : H.ISFL

Appel : routine ISFLIO (« IS FiLe I/O »), test d'existence d'un fichier

Fonct. : permet de détourner la routine

0FEE4H **H.OUTD**

Nom : H.OUTD

Appel : routine OUTDO (« OUT DO »), sortie d'un caractère sur écran ou imprimante

Fonct. : permet de détourner la routine

0FEE9H **H.CRDO**

Nom : H.CRDO

Appel : routine CRDO (« CRlf DO »), émission d'un CR (carriage return - code 0DH) puis d'un LF (line feed - code 0AH)

Fonct. : permet, par exemple, d'adapter les sorties sur une imprimante possédant un line-feed automatique

0FEEEH **H.DSKC**

Nom : H.DSKC

Appel : routine DSKCHI (« DiSK CHaracter Input »)

Fonct. : permet de détourner la routine

0FEF3H **H.DOGR**

Nom : H.DOGR

Appel : routine DOGRPH (« DO GRaPH »), utilisée par les fonctions graphiques du Basic

Fonct. : permet de détourner la routine

0FEF8H **H.PRGE**

Nom : H.PRGE

Appel : routine PRGEND (« PRoGram END »), appelée à la fin de l'exécution d'un programme Basic

Fonct. : permet n'importe quelle action avant de rendre la main à l'utilisateur

0FEFDH **H.ERRP**

Nom : H.ERRP

Appel : routine ERRPRT (« ERRor PRinT »), utilisée lors de l'affichage par le Basic d'un message d'erreur

Fonct. : permet de traiter les erreurs directement

0FF02H **H.ERRF**

Nom : H.ERRF

Appel : à la fin de la routine affichant un message d'erreur sous Basic

Fonct. : permet une action supplémentaire après une erreur

0FF07H **H.READ**

Nom : H.READ

Appel : routine READY, utilisée à chaque fois que le message « Ok » (ou le prompt défini par l'utilisateur sur MSX2) est affiché à l'écran

Fonct. : permet de faire beaucoup de choses, par exemple provoquer l'émission d'un bref bip sonore à chaque affichage de « Ok »

0FF0CH **H.MAIN**

Nom : H.MAIN

Appel : routine MAIN (« MAIN entry »), utilisée à chaque accès à l'interpréteur Basic (autrement dit : en permanence sous Basic)

Fonct. : permet de programmer une interruption par exemple

0FF11H H.DIRD

Nom : H.DIRD

Appel : routine DIRDO (« DIRect DO »), utilisé lors de l'exécution d'une instruction en mode direct

Fonct. : permet toutes les manipulations sur le mode direct du Basic

0FF16H H.FINI

Nom : H.FINI

Appel : routine FININT (« Function INterpretation INiTialize »), utilisée lors de la traduction d'une instruction Basic par l'interpréteur

Fonct. : permet de détourner le traitement des instructions Basic

0FF1BH H.FINE

Nom : H.FINE

Appel : routine FINEND (« Function INterpretation END »), utilisée à la fin de l'interprétation d'une instruction Basic

Fonct. : permet de détourner la routine

0FF20H H.CRUN

Nom : H.CRUN

Appel : routine CRUNCH, transformation d'une ligne Basic en mots-clefs

Fonct. : permet de détourner la routine

0FF25H H.CRUS

Nom : H.CRUS

Appel : routine CRUSH, recherche d'un mot-clef dans la liste alphabétique

Fonct. : permet de détourner la routine

0FF2AH H.ISRE

Nom : H.ISRE

Appel : routine ISRESV (« IS token RESerVed »), découverte d'un mot-clef lors de l'exécution de la routine CRUNCH

Fonct. : permet de détourner la routine

0FF2FH **H.NTFN**

Nom : H.NTFN

Appel : routine NTFN2, utilisée lorsqu'un mot-clef est suivi par un numéro de ligne

Fonct. : permet de détourner la routine

0FF34H **H.NOTR**

Nom : H.NOTR

Appel : routine NOTRSV (« token NOT ReSerVed »), la suite de caractères examinée ne constitue pas un mot-clef

Fonct. : permet de détourner la routine

0FF39H **H.SNGF**

Nom : H.SNGF

Appel : routine SNGFOR, accès aux fonctions mathématiques BCD

Fonct. : permet d'installer de nouvelles routines mathématiques

0FF3EH **H.NEWS**

Nom : H.NEWS

Appel : routine NEWSTT(« NEW STaTement »), passage à une nouvelle instruction Basic

Fonct. : permet de détourner la routine

0FF43H **H.GONE**

Nom : H.GONE

Appel : routine GONE2, instruction de déroutement en cours

Fonct. : permet de détourner la routine

0FF48H **H.CHRG**

Nom : H.CHRG

Appel : routine CHRGET (« CHaRacter GET »), saisie d'un caractère au clavier

Fonct. : permet de détourner la routine

0FF4DH **H.RETU**

Nom : H.RETU

Appel : routine RETURN (« RETURN from subroutine ») utilisée lors de l'instruction Basic RETURN

Fonct. : permet de détourner la routine

0FF52H **H.PRTF**

Nom : H.PRTF

Appel : routine PRTFLD, utilisée lors de l'instruction Basic PRINT

Fonct. : permet de détourner la routine

0FF57H **H.COMP**

Nom : H.COMP

Appel : routine COMPRT, boucle interne à l'instruction Basic PRINT

Fonct. : permet de détourner la routine

0FF5CH **H.FINP**

Nom : H.FINP

Appel : routine FINPRT (« FINal PRinT »), fin de l'instruction Basic PRINT

Fonct. : permet de détourner la routine

0FF61H **H.TRMN**

Nom : H.TRMN

Appel : routine TRMNOK, traitement d'une donnée incorrecte lors des instructions Basic DATA et INPUT

Fonct. : permet de détourner la routine

0FF66H **H.FRME**

Nom : H.FRME

Appel : routine FRMEVL (« FoRMula EVaLuator »), calcul du résultat d'une formule mathématique

Fonct. : permet d'installer de nouvelles routines mathématiques

0FF6BH **H.NTPL**

Nom : H.NTPL

Appel : routine utilisée lors de FRMEVL

Fonct. : permet d'installer de nouvelles routines mathématiques

0FF70H **H.EVAL**

Nom : H.EVAL

Appel : routine EVALST (« EVALuate STatement »), évaluation d'une expression

Fonct. : permet de détourner la routine

0FF75H **H.OKNO**

Nom : H.OKNO

Appel : routine de calcul de fonctions transcendantes

Fonct. : permet de détourner la routine

0FF7AH **H.FING**

Nom : H.FING

Appel : à la fin de la routine de calcul de fonctions transcendantes

Fonct. : permet de détourner la routine

0FF7FH **H.ISMI**

Nom : H.ISMI

Appel : routine ISMID$ (« IS MID$ »), utilisée lors de l'instruction MID$ du Basic

Fonct. : permet de détourner la routine

0FF84H **H.WIDT**

Nom : H.WIDT

Appel : routine WIDTHS (« WIDTH of Screen »), utilisée lors de l'instruction Basic WIDTH

Fonct. : permet de détourner la routine

0FF89H **H.LIST**

Nom : H.LIST

Appel : routine LIST , utilisée lors des instructions Basic LIST et LLIST

Fonct. : permet de détourner la routine

0FF8EH **H.BUFL**

Nom : H.BUFL

Appel : routine BUFLIN (« BUFfer LINe »), utilisée lors de l'instruction Basic LIST

Fonct. : permet de détourner la routine

0FF93H **H.FRQI**

Nom : H.FRQI

Appel : routine FRQINT, utilisée lors de l'instruction Basic POKE

Fonct. : permet de détourner la routine

0FF98H **H.SCNE**

Nom : H.SCNE

Appel : routine SCNEX2, conversion d'un numéro de ligne en adresse mémoire et inversement

Fonct. : permet de détourner la routine

0FF9DH **H.FRET**

Nom : H.FRET

Appel : routine FRETMP (« FREe up TeMPories »), recherche d'un emplacement libre pour stocker une variable alphanumérique

Fonct. : permet de détourner la routine

0FFA2H **H.PTRG**

Nom : H.PTRG

Appel : routine PTRGET (« PoinTeR GET »)

Fonct. : permet d'utiliser d'autres noms de variables

0FFA7H **H.PHYD**

Nom : H.PHYD

Appel : routine PHYDIO (« PHYsical Disk I/O ») du Bios, adresse 00144H en main-ROM

Fonct. : utilisé par le système d'exploitation des disquettes

0FFACH **H.FORM**

Nom : H.FORM

Appel : routine FORMAT (« disk FORMATter ») du Bios, adresse 00147H en main-ROM

Fonct. : utilisé par le système d'exploitation des disquettes

0FFB1H **H.ERRO**

Nom : H.ERRO

Appel : routine ERROR, traitement d'une erreur sous Basic

Fonct. : permet de traiter automatiquement une erreur

0FFB6H **H.LPTO**

Nom :	H.LPTO
Appel :	routine LPTOUT (« Line PrinTer OUTput »), sortie d'un caractère sur imprimante
Fonct. :	permet d'utiliser une imprimante non-MSX

0FFBBH **H.LPTS**

Nom :	H.LPTS
Appel :	routine LPTSTT (« Line PrinTer STaTus »), test de l'état de l'imprimante
Fonct. :	permet d'utiliser une imprimante non-MSX

0FFC0H **H.SCRE**

Nom :	H.SCRE
Appel :	routine SCREEN (« set SCREEN »), utilisée lors de l'instruction Basic SCREEN
Fonct. :	permet de détourner la routine

0FFC5H **H.PLAY**

Nom :	H.PLAY
Appel :	routine PLAY, utilisée lors de l'instruction Basic PLAY
Fonct. :	permet de détourner la routine

5 LE PROCESSEUR GRAPHIQUE (V9938)

5.1 AVERTISSEMENT

Avec le processeur graphique, nous abordons un domaine plus spécifique au MSX2. Les possesseurs de MSX1 trouveront néanmoins tous les renseignements nécessaires pour faire fonctionner correctement le processeur vidéo du MSX1 (V9929) en ne tenant compte que des modes d'écran du MSX1 (SCREEN 0, 1, 2 et 3). Quant aux registres, ils seront détaillés aussi bien dans le cas du MSX1 que du MSX2.

5.2 INTRODUCTION AU V9938

Le processeur vidéo V9938, ou VDP, pour Video Display Processor, est sans aucun doute l'élément le plus excitant du système MSX2. Avec sa mémoire vidéo énorme (128 Ko), il offre une résolution graphique extraordinaire pour du matériel grand public, puisqu'il autorise par exemple l'afichage de deux cent cinquante six couleurs simultanément à l'écran. Il comporte par ailleurs un jeu complet d'instructions graphiques permettant notamment le tracé de lignes, de rectangles pleins ou vides, de

scrolling pour ne citer que quelques fonctions. La puissance du V9938 vient du fait que c'est un processeur VLSI (Very Large Scale Integration), c'est-à-dire qu'il atteint un degré d'intégration encore inégalé en micro-informatique. Le V9938 restera dans l'histoire comme le premier processeur VLSI présent en informatique grand public.

5.3 FONCTIONNEMENT DU V9938

Le principe d'un processeur graphique est relativement simple. Il s'agit de soulager le micro-processeur central de toutes les tâches liées au graphisme en lui adjoignant un second processeur dédié à l'affichage. Au niveau de la programmation, on peut ignorer totalement la présence du processeur et n'utiliser que les routines du Bios (tracé de lignes, chargement de pages, sprites, etc). Cette solution, si elle offre l'avantage de la simplicité, prive le programmeur de toute une série de fonctions non-disponibles avec le Bios (scrolling, clignotement pour ne citer que deux exemples). Il est donc intéressant de pouvoir accéder directement au processeur graphique. Ceci s'opère par l'intermédiaire de registres, huit sur MSX1 et quarante sept sur MSX2 - ce qui vous donne une idée de la différence de puissance entre le processeur du MSX1 et celui du MSX2. La plupart des registres définissent des paramètres précis, la couleur du texte par exemple, alors que quelques registres ont une fonction spéciale, écrire dans le registre 46 déclenche automatiquement une opération, le tracé d'une ligne par exemple. Vous trouverez tout au long de ce chapitre l'explication du contenu de chaque registre ainsi qu'un exposé des différents modes graphiques et des sprites.

5.4 COMMENT ACCEDER AUX REGISTRES DU V9938

Il existe deux moyens d'accéder au processeur vidéo. Tout dépend de ce que vous cherchez à faire :

- soit vous avez le temps et vous voulez simplement mettre en œuvre les fonctions inexistantes sous Bios.
- soit vous avez décidé d'utiliser directement le V9938 pour une application qui nécessite la vitesse la plus élevée possible. L'utilisation du Bios ralentit légèrement l'application d'un programme.

Dans le premier cas, la solution est évidente : utilisez les routines WRTVDP, RDVDP, SETPLT, GETPLT et VDPSTA du Bios. Dans le second cas, la situation est moins simple. Il est évident que si vous voulez éviter d'appeler le Bios pour les fonctions graphiques, vous n'allez pas l'appeler pour écrire dans les registres. Or, hors du Bios, point de salut, vous perdez la compatibilité dès que vous utilisez l'instruction OUT du Z80. Heureusement, Microsoft a prévu ce cas de figure lors de la conception du système MSX. Voici la marche à suivre :

ECRIRE DANS UN REGISTRE DE CONTROLE (0-23 et 32-46)

- lire le contenu de la case mémoire 7 en main-ROM (si vous ne comprenez pas la signification de main-ROM, voyez le chapitre concernant les slots)
- vous aurez l'adresse du port zéro du processeur vidéo. Or nous utiliserons le port un pour écrire dans le VDP, une incrémentation d'impose.
- envoyez sur le port un la donnée à écrire (« OUT »).
- envoyez, toujours sur le port un, le numéro du registre dans lequel vous voulez écrire en gardant le bit de poids fort à 1, ce qui revient à ajouter 80H au numéro du registre.

Un exemple :

< écrire 0 dans le registre neuf du V9938 >

```
                ORG   0C000H
;
; Le programme suivant change la fréquence d'affichage en passant de 50 à 60 Hertz
; Sur un téléviseur ou un moniteur en 50 Hz uniquement, l'image « sautera ».
; Il faut remettre 2 dans le registre neuf afin de rétablir une image normale.
DEBUT :         LD    A, (7)        ; adresse port 0 dans A
                LD    C, A          ; adresse port 0 dans C
                INC   C             ; adresse port 1 dans C
;
                LD    A, 0          ; donnée dans A
                OUT   (C), A        ; envoi sur port 1
;
                LD    A, 80H+9      ; n° registre + 80H dans A
                OUT   (C), A        ; envoi sur port 1
                RET
;
; NOTE : ce programme ne « masque » pas les bits qui ne nous intéressent pas
; (pour être plus simple), il est possible qu'il ne marche pas correctement sur certains
; MSX2.
;
                END   DEBUT
```

L'équivalent Basic de ce programme serait :

```
C=PEEK(7) : C=C+1 : OUT C,2 : OUT C, &H80+9
```

Dans certains cas, il peut être utile de charger les registres à la suite ou de charger plusieurs fois le même registre. L'accès indirect est possible :

- chargez le registre 17 du processeur avec le numéro du registre dans lequel vous désirez écrire avec la méthode précédente
- envoyez vos données, les unes à la suite des autres sur le port 3 du processeur vidéo

L'opération d'auto-incrémentation a normalement lieu, à savoir que le numéro du registre dans le registre 17 augmente à chaque fois. Il est possible d'annuler cette fonction de manière à toujours écrire dans le même registre. Il suffit de mettre à 1 le bit de poids fort du registre 17, ce qui revient à ajouter 80H.

Un second exemple :

< écrire 0 dans le registre 9 >

< attendre un peu >

< écrire 2 dans le registre 9 >

```
                ORG   0C000H
DEBUT:          LD    A, (7)
                LD    C, A
                INC   C
```

```
                LD    A, 80H+9        ; registre 9 sans auto-incrémentation
                OUT   (C), A
                LD    A, 80H+17       ; dans registre 17 (+80H)
                OUT   (C), A
;
                INC   C               ; port 2 dans C
                INC   C               ; port 3 dans C
                LD    A, 0            ; 0=60 Hz
                OUT   (C),A           ; sur port 3
;
                CALL  ATTEND          ; boucle d'attente
;
ATTEND:         PUSH  BC              ; sous-programme de
                LD    BC, 0           ; boucle d'attente
BCL:            DEC   BC              ; pour voir un peu
                LD    A, B            ; quelque chose
                OR    C
                CP    0
                JR    NZ, BCL
                POP   BC
                RET
```

ECRIRE DANS UN REGISTRE DE PALETTE :

- chargez le registre 16 avec le numéro de la couleur à modifier (0-15) par une des deux manières ci-dessus
- envoyez sur le port deux du processeur vidéo, deux octets qui codent la nouvelle teinte :

```
octet un :    0   b6   b5   b4   0   b2   b1   b0
                  !    !    !        !    !    !
                  !    !    !        +----+----+--  bleu
                  !    !    !
                  +----+----+-----------------------  rouge
octet deux :  0   0    0    0    0   b2   b1   b0
                                     !    !    !
                                     +----+----+-- vert
```

Exemple :

< changer la couleur n°1 (noir), en vert >

```
                ORG   0C000H
        DEBUT:  LD    A, (7)          ; port 0 dans A
                LD    C, A            ; port 0 dans C
                INC   C               ; port 1 dans C
        ;
                LD    A, 1            ; couleur 1
                OUT   (C), A
                LD    A, 80H+16
                OUT   (C), A          ; dans le registre 16
        ;
                INC   C               ; port 2 dans C
                LD    A, 42H          ; rouge=4 bleu =2
                OUT   (C), A          ; et hop...
```

```
                LD      A, 5                ; vert=5
                OUT     (C), A              ; et re-hop...
;
                RET
                END     DEBUT
```

LIRE UN REGISTRE D'ETAT (STATUS REGISTER) :

- charger le registre 15 du processeur vidéo avec le numéro de registre status que vous désirez lire (0-9).
- lire le contenu du registre sur le port 1.

Exemple :

< lire le registre status 0 >

```
                ORG     0C000H
;
DEBUT:          LD      A, (7)              ; port 0 dans A
                LD      C, A                ; port 0 dans C
                INC     C                   ; port 1 dans C
;
                LD      A, 0                ; le registre...
                OUT     (C), A              ; ... status n°0
                LD      A, 80H+15
                OUT     (C), A              ; dans le registre 15
;
                LD      A, 7                ; port 0 dans A
                LD      C, A                ; port 0 dans C
                INC     C                   ; port 1 dans C
;
                IN      (C), A              ; contenu dans A
;
                RET
```

5.5 LECTURE ET ECRITURE DANS LA MEMOIRE VIDEO

Un MSX2 peut être équipé de 64, 128 ou 192 Ko de mémoire vive réservée à la vidéo (ou VRAM). Tous les modèles commercialisés en France comportent 128 Ko. Cette mémoire ne se trouve pas dans un slot, ce faisant, le Z80 ne peut en aucun cas l'adresser, seul le processeur vidéo peut y accéder.

– carte mémoire vidéo -

–

```
***********************     ---- 1FFFFH ----
*                     *
*       128 Ko        *
*                     *
***********************     ---- 0FFFFH ----     ***********************
*                     *                          *                     *
*        64 Ko        *                          *       192 Ko        *
*                     *                          *                     *
***********************     ---- 00000H ----     ***********************

    VRAM principale                                   VRAM étendue
     - affichage -                                    - données -
```

Il existe trois méthodes permettant de manipuler la mémoire vidéo. La première consiste à utiliser le Bios, puisqu'on y trouve les routines WRTVRM, RDVRM qui respectivement écrivent et lisent la mémoire vidéo. Les deux autres méthodes nécessitent des accès directs au processeur vidéo, l'une passe par les registres alors que l'autre passe par les commandes du processeur vidéo. pour ce qui est de cette dernière solution, elle est détaillée dans la partie concernant les commandes propres au VDP. Nous allons à présent examiner l'accès au processeur vidéo grâce aux registres.

- choisir la VRAM principale ou la VRAM secondaire en agissant sur le bit 6 du registre 45 du VDP (0=VRAM principale, 1=VRAM étendue). Cette opération n'est nécessaire que lors du premier accès vidéo à moins que vous n'ayez 192 Ko de VRAM.
- toute adresse est codée sur 17 bits (0 à 1FFFFH). Charger le registre 14 avec les 3 bits de poids fort de l'adresse qui vous intéresse (A16, A15 et A14).
- envoyer sur le port 1 du VDP les 8 bits de poids faible de l'adresse (bits A7 à A0)
- envoyer sur le port 1 du VDP les bits manquants de l'adresse (bits A13 à A8) ainsi que l'instruction d'écriture ou de lecture.

```
octet          b7   b6   b5   b4   b3   b2   b1   b0
               !    !    !    !    !    !    !    !
               !    !    !    !    !    !    !    +-- bit A8
               !    !    !    !    !    !    +----------- bit A9
               !    !    !    !    !    +------------------- bit A10
               !    !    !    !    +---------------------------- bit A11
               !    !    !    +------------------------------------ bit A12
               !    !    +-------------------------------------------- bit A13
               !    !
               +-------+---------------------------------------------------- 0=lecture, 1=écriture
```

- envoyer la donnée sur le port 0 du VDP. L'auto-incrémentation ayant lieu, il n'est pas nécessaire de redéfinir l'adresse à chaque accès dans le cas de blocs par exemple.

Exemple :

< en SCREEN 0 / 40 colonnes, envoyer des caractères dans la table des noms afin d'afficher

un message en haut à droite de l'écran >

```
            ORG   0C000H
;
DEBUT:      LD    A, 7
            LD    C, A
            INC   C
            LD    A, 0              ; 3 bits de poids...
            OUT   (C), A            ; ... fort dans...
            LD    A, 80H+14
            OUT   (C), A            ; ... registre 14
;
            LD    A, 01FH           ; 8 bits de poids...
            OUT   (C), A            ; ... faible = 31
;
            LD    A, 040H           ; écriture avec...
            OUT   (C), A            ; autres bits à 0
;
            DEC   C                 ; port 0 dans C
            LD    HL, DATA
            LD    B, 9              ; neuf lettres
            OTIR                    ; transfert auto-incrémenté
            RET
DATA:       DB    'CA MARCHE'
            END   DEBUT
```

5.6 LES REGISTRES « WRITE ONLY » DU PROCESSEUR VIDEO V9938

Voici la liste complète de tous les registres accessibles en écriture à l'utilisateur et la signification des informations qu'ils contiennent.

Registre 0

0	DG	IE2	IE1	M5	M4	M3	EV

MSX1/MSX2

DG - (réservé MSX2) 1 pour mettre le bus de couleur en mode « entrée » et récupérer les données en VRAM

IE2 - (réservé MSX2) 1 pour autoriser les interruptions du crayon optique

IE1 - (réservé MSX2) 1 pour autoriser les interruptions du scan horizontal

M5 - (réservé MSX2) bit de mode graphique

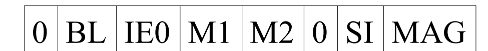
M4 - (réservé MSX2) bit de mode graphique

M3 - (MSX1 et MSX2) bit de mode graphique

EV - (réservé MSX1) 1 pour une entrée vidéo externe, 0 interdit l'entrée vidéo externe

Note : pour plus de détails sur les bits de mode graphique, voir le registre 1 du VDP ci-dessous.

Registre 1



0	BL	IE0	M1	M2	0	SI	MAG

MSX1/MSX2

BL - (MSX1 et MSX2) 1 pour allumer l'écran «(« Screen display enable »), 0 pour éteindre l'écran sauf la marge

IE0 - (MSX1 et MSX2) autorise les interruptions du VDP

M1 - (MSX1 et MSX2) bit de mode graphique

M2 - (MSX1 et MSX2) bit de mode graphique

SI - (MSX1 et MSX2) bit de taille des sprites (« size ») : 1 pour les sprites 16x16, 0 pour les sprites 8x8

MAG - taille des sprites : 1 pour les sprites doubles, 0 pour les sprites ordinaires

note : le mode graphique est déterminé par les bits M1 à M5 de la manière suivante :

	M5	M4	M3	M2	M1	
SCREEN 0	0	0	0	0	1	40 colonnes
SCREEN 0	0	1	0	0	1	80 colonnes
SCREEN 1	0	0	0	0	0	32 colonnes
SCREEN 2	0	0	1	0	0	256x192, 16 couleurs
SCREEN 3	0	0	0	1	0	64x48, 16 couleurs
SCREEN 4	0	1	0	0	0	idem SCREEN 2 + 8 sprites
SCREEN 5	0	1	1	0	0	256x212 bitmap, 16 couleurs
SCREEN 6	1	0	0	0	0	512x212 bitmap, 4 couleurs
SCREEN 7	1	0	1	0	0	512x212 bitmap, 16 couleurs
SCREEN 8	1	1	1	0	0	256x212 bitmap 256 couleurs

Registre 2	0	N16	N15	N14	N13	N12	N11	N10	MSX1/MSX2

Sur MSX1 :

N14 à N16 n'existent pas

N10 à N13 codent les quatre bits de poids fort de l'adresse du début de la table des noms en VRAM. L'adresse véritable s'obtient en multipliant la valeur de ces 4 bits par 400H. Par exemple, si les bits sont 1001 (soit 9 en hexa), la table des noms se trouve en 9*400H=2400H.

L'adresse peut varier dans ces conditions entre 0 et 3C00H

Sur MSX2 :

N10 à N16 codent les 7 bits de poids fort de l'adresse du début de la table des noms. Le fonctionnement est le même que sur MSX1 (voir ci-dessus). L'adresse varie entre 0 et 1FC00H.

Registre 3

C13	C12	C11	C10	C9	C8	C7	C6

MSX1/MSX2

Sur MSX1 :

C6 à C13 codent les 8 bits de poids fort de l'adresse du début de la table des couleurs. L'adresse réelle en mémoire s'obtient donc en multipliant le contenu du registre 3 par 40H. Par exemple si l'on lit 0B6H dans le registre 3, l'adresse de début de la table des couleurs est 0B6H*40H=2D80H.

L'adresse peut varier entre 0 et 3FC0H.

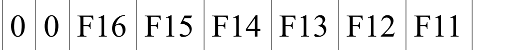
ATTENTION : en SCREEN 2, le fonctionnement est différent, la table des couleurs ne peut se trouver qu'en 0 ou 2000H. Seul le bit de poids fort intervient. Les autres bits sont tous à 1. Le registre 3 ne devra donc contenir que 7FH ou FFH.

Sur MSX2 :

C6 à C13 s'utilisent avec C14, C15 et C16 du registre 10 du V9938. Ces 11 bits codent l'adresse du début de la table des couleurs comme sur MSX1 (voir ci-dessus).

L'adresse peut varier entre 0 et 1FFC0H.

Registre 4



0	0	F16	F15	F14	F13	F12	F11

MSX1/MSX2

Sur MSX1 :

F14 à F16 n'existent pas.

F11 à F13 codent les 3 bits de poids fort de l'adresse du début de la table des formes en VRAM. L'adresse véritable s'obtient donc en multipliant la valeur de ces 3 bits par 800H. Par exemple, si les bits sont 100 (soit 4 en hexa), la table des formes se trouve en 4*800H=2000H.

L'adresse peut varier dans ces conditions entre 0 et 3800H.

ATTENTION : en SCREEN 2, le fonctionnement est différent. La table des formes ne peut commencer qu'en 0 ou 2000H. Seul le bit 2 du registre 4 intervient. Les 2 bits de poids faible sont toujours à 1. Les deux seules valeurs possibles sont donc 03H et 07H. De surcroît, le bit 2 du registre 4 doit toujours être positionné à l'inverse du bit 8 du registre 3. (Vous me suivez bien ?)

Sur MSX2 :

F11 à F16 codent les 6 bits de poids fort de l'adresse du début de la table des formes. Le fonctionnement est le même que sur MSX1 (voir ci-dessus). L'adresse varie entre 0 et 1F800H.

Registre 5

S14	S13	S12	S11	S10	S9	S8	S7

MSX1/MSX2

Sur MSX1 :

S14 n'existe pas

S7 à S13 codent les 7 bits de poids fort de l'adresse du début de la table des attributs de sprites. L'adresse réelle s'obtient donc en multipliant le contenu du registre 5 par 80H. Par exemple si l'on lit 037H dans le registre 5, l'adresse du début de la table des attributs de sprites est 037H*80H=1B80H.

L'adresse peut varier entre 0 et 3F80H.

Sur MSX2 :

S7 à S14 s'utilisent avec S15 et S16 du registre 11 du V9938. Ces 10 bits codent l'adresse du début de la table des attributs de sprites comme sur MSX1 (voir ci-dessus).

L'adresse peut varier entre 0 et 1FF80H.

Registre 6

0	0	P16	P15	P14	P13	P12	P11

MSX1/MSX2

Sur MSX1 :

P14 à P16 n'existent pas.

P11 à P13 codent les 3 bits de poids fort de l'adresse du début de la table de génération des sprites en VRAM. L'adresse véritable s'obtient donc en multipliant la valeur de ces 3 bits par 800H. Par exemple, si les bits sont 100 (soit 4 en hexa), la table des formes de sprites se trouve en 04H*800H=2000H.

L'adresse peut varier dans ces conditions entre 0 et 3800H.

Sur MSX2 :

P11 à P16 codent les 6 bits de poids fort e l'adresse du début de la table de génération des sprites. Le fonctionnement est le même que sur MSX1 (voir ci-dessus). L'adresse varie entre 0 et 1F800H.

Registre 7

TC3	TC2	TC1	TC0	BD3	BD2	BD1	BD0

MSX1/MSX2

TC0 à TC3 donnent la couleur du texte dans les modes textes.

BD0 à BD3 donnent la couleur du fond dans tous les SCREEN.

Note : VDP(7)=&HA0, par exemple, fait apparaître un texte jaune sur un fond transparent (noir), mais ce registre a une autre utilisation, voir les registres 12 et 13.

Registre 8

MS	LP	TP	CB	VR	0	SPD	BW

MSX2

MS - mis à 1 lorsqu'on utilise une souris, reste à 0 dans le cas contraire (« MouSe enable »).

LP - idem pour le crayon optique (« Light Pen enable »).

TP - 0 = couleur 0 transparent, 1 = couleur 0 redéfinissable.

CB - bit du bus de couleur (« Color Bus ») 1 en entrée, 0 en sortie

VR - donne le type de mémoire vidéo (« Video Ram ») :

1 = 64Ko * 1 bit ou 64Ko * 4 bits

0 = 16 Ko * 1 bit ou 16 Ko * 4 bits

SPD - affichage des sprites (« SPrites Display »)

1 = sprites éteints

0 = sprites affichés

BW - choix de la couleur ou du noir et blanc (« Black and White »)

1 = affichage en noir et blanc (32 nuances de gris)

0 = affichage en couleur

Le bit TP indique si la couleur n°0 est utilisée comme une couleur normale (TP à 1), à savoir redéfinissable par exemple avec l'instruction COLOR = (0, R, G, B). Dans le cas contraire, (TP à 0), la couleur n°0 est « transparente ». Il est possible alors d'avoir un fond en vidéo par exemple (utilisé lors de l'incrustation vidéo). En Basic, on modifie l'état de TP par :

VDP(9) = VDP(9) OR &H20 pour mettre TP à 1

VDP(9) = VDP(9) AND &HDF pour mettre TP à 0

Registre 9

LN	0	S1	S0	IL	E0	NT	DC

MSX2

LN - 1 = hauteur de l'écran réglée à 212 points

0 = hauteur de l'écran réglée à 192 points

S0 et S1 - 0 = normal, 1 = digitalisation, incrustation, etc, 2 = vidéo externe

IL - 1 = affichage entrelacé (NTSC)

0 = affichage non-entrelacé

E0 - 1 = alternance de deux écrans en mode graphique

0 = pas d'alternance

Pour plus de précisions, voir le registre 13

NT - 1 = affichage en PAL (313 lignes), 50 Hertz

0 = affichage en NTSC (256 lignes), 60 Hertz

DC - 1 = DTCLK en mode entrée

0 = DTCLK en mode sortie

Registre 10

0	0	0	0	0	C16	C15	C14

MSX2

C14 à C16 sont les 3 bits de poids fort de l'adresse du début de la table des couleurs (qui en comporte 17). Voir le registre 3 pour plus de précisions.

Registre 11

0	0	0	0	0	0	S16	S15

MSX2

S15 et S16 sont les 2 bits de poids fort de l'adresse du début de la table des attributs de sprites (qui en comporte 17). Voir le registre 5 pour plus de précisions.

Registre 12

T23	T22	T21	T20	BC3	BC2	BC1	BC0

MSX2

Lors d'un clignotement, le texte s'affiche dans les couleurs codées par ce registre pendant la période ON.

T20 à T23 seconde couleur du texte lors d'un clignotement en mode texte (80 colonnes uniquement). Voir le registre 13 pour plus de précisions.

BC0 à BC3 seconde couleur de fond lors d'un clignotement en mode texte (80 colonnes uniquement). Voir registre 13

Registre 13

ON3	ON2	ON1	ON0	OF3	OF2	OF1	OF0

MSX2

Ce registre permet d'obtenir automatiquement l'affichage alterné de deux pages graphiques (SCREEN 5 à 8) ou le clignotement automatique du texte (SCREEN 0, 80 colonnes).

ON0 à ON3 règle le temps d'affichage de la première page (page paire) ou de la seconde couleur de texte.

OF0 à OF3 règle le temps d'affichage de la seconde page (page impaire) ou de la première couleur de texte.

Note : longueur d'une période en secondes pour une fréquence de 50 Hz :

0000 = 0,0	0100 = 0,8	1000 = 1,6	1100 = 2,4
0001 = 0,2	0101 = 1,0	1001 = 1,8	1101 = 2,6
0010 = 0,4	0110 = 1,2	1010 = 2,0	1110 = 2,8
0011 = 0,6	0111 = 1,4	1011 = 2,2	1111 = 3,0

Utilisation en mode texte 80 colonnes (SCREEN 0) :

- charger le registre 7 avec les couleurs du texte et du fond pour tout l'écran.
- charger le registre 12 avec les couleurs de texte et de fond à afficher en alternance pour le texte qui clignote.
- Modifier la table des couleurs en VRAM. Cette table occupe 240 octets (800Hà 8EFH par défaut), sachant que le MSX2 utilise 1 bit par caractère (24 lignes x 80 colonnes = 1920 caractères d'où 1920 bits, soit 240 octets). Si le bit est à 1, le texte clignote (alternance des couleurs contenues dans les registres 7 et 12) alors que s'il se trouve à 0, le caractère ne clignote pas.
- Régler le registre 13 avec les temps d'affichage adéquats.

Voici un exemple en Basic :

```
10 SCREEN 0 : WIDTH80
20 COLOR 10, 0, 0 : FOR I = &H800+239 : VPOKE I,0 : NEXT
30 C = PEEK(7) : C = C+1 : OUT C,&H10 : OUT C,&H80+12 : OUT C,&H44 : OUT C,&H80+13
: CLS
40 PRINT « Ce petit programme permet de faireclignoter automatiquement un texte ! »: PRINT :
PRINT : VPOKE &H804, &H1F : VPOKE &H805, &HF0
```

Après avoir lancé le programme ci-dessus, vous pouvez continuer à travailler. Essayez de changer le &H44 de la ligne 30 en &H11 et refaites RUN. Vous avez changé la vitesse dans le registre 13. Mettez maintenant &HBB à la place de &H11, en même temps, changez le &H10 en &H67. Le mot apparaît à présent en jaune sur noir (comme le reste) puis en rouge sur fond cyan. Vous avez agi sur le registre 12. Enfin, transformez le VPOKE &H805,&HF0 en VPOKE &H805,&HFF. A l'exécution, vous observerez que vous avez modifié des flags et que de nouveaux caractères clignotent.

Utilisation en mode graphique (SCREEN 5 à 8) :

Les pages graphiques vont par deux. En SCREEN 5 et 6, on peut choisir de faire afficher en alternance les pages 0 et 1 ou alors 2 et 3 mais jamais 0 et 3, 1 et 3 ou 0 et 2..........................

- Mettre les bits N10 à N14 du registre 2 à 1, puis choisir la page impaire avec les bits N15 et N16. En SCREEN 7 et 8, ce ne peut être que la page 1. En SCREEN 5 et 6, on peut choisir la page 1 ou 3 (il faut charger le registre 2 avec respectivement 3FH ou 7FH).
- Régler les temps d'affichages respectifs grâce au registre 13.
- Mettre le bit E0 du registre 9 à 1.

Attention, il est absolument nécessaire d'être sur la page paire lorsqu'on démarre ce type de manipulation depuis le Basic.

Voici justement un exemple en Basic :

```
5 SCREEN 5 : PI = 3.141592654#
10 SET PAGE 0, 0 : COLOR 10, 0, 0 : CLS
20 CIRCLE (100, 100), 50, 7, PI/2, 2*PI
30 LINE (50, 100)-(150, 100), 7 : LINE (100, 50)-(100, 150), 7 : PAINT (98, 98), 10, 7 : PAINT
(102, 102), 6, 7: PAINT(98, 102), 12, 7
40 SET PAGE 1, 1 : CIRCLE (100, 100), 50, 7, PI/4, 2*PI-PI/4
50 LINE (65, 65)-(135, 135), 7 : LINE (65, 135)-(135, 65), 7 : PAINT (100, 98), 10, 7 : PAINT
(100, 102), 6, 7: PAINT(98, 100), 12, 7
85 '
90 SET PAGE 0, 0 : C = PEEK(7) : C = C+1 : OUT C, &H3F : OUT C,&H82 : OUT C, &H22 :
OUT C,&H8D : OUT C, 6 : OUT C, &H89
95 '
99 GOTO 99
```

Les lignes 10 à 60 se chargent de faire un joli dessin. La ligne 90 démarre l'affichage alterné automatique.

Registre 14	0	0	0	0	0	V16	V15	V14	MSX2

V14 à V16 codent les 3 bits de poids fort de l'adresse VRAM à laquelle on désire accéder. Voir le paragraphe 5.5 « Lecture et écriture dans la mémoire vidéo ».

Registre 15	0	0	0	0	ST3	ST2	ST1	ST0	MSX2

ST0 à ST3 contiennent le numéro de registre d'état que l'on désire lire (0-9). Voir le paragraphe 5.4 « Comment accéder aux registres du V9938 » rubrique « LIRE UN REGISTRE D'ETAT (STATUS REGISTER) », pour plus de précisions.

Registre 16	0	0	0	0	CC3	CC2	CC1	CC0	MSX2

CC0 à CC3 permettent de définir le numéro de la couleur lors d'un accès palette. Voir le paragraphe 5.4 « Comment accéder aux registres du V9938 » rubrique « ECRIRE DANS UN REGISTRE DE PALETTE », pour plus de précisions.

Registre 17	AII	0	RS5	RS4	RS3	RS2	RS1	RS0	MSX2

Ce registre est utilisé lors d'un accès indirect à un autre registre du V9938

AII - flag d'auto-incrémentation : 1 = auto-incrémentation

0 = mode normal

RS0 à RS5 - donnent le numéro de registre du V9938 désiré. Voir le paragraphe 5.4 « Comment accéder aux registres du V9938 » rubrique « ECRIRE DANS UN REGISTRE DE CONTROLE », pour plus de précisions.

Registre 18	V3	V2	V1	V0	H3	H2	H1	H0	MSX2

H0 à H3 - correction horizontale de l'affichage (SET ADJUST du Basic)

7 - 6 - 5 - ... - 1 - 0 - 15 - 14 - ... - 10 - 9 - 8

gauche centre droite

V0 à V3 - correction verticale de l'affichage (SET ADJUST du Basic)

8 - 7 - 6 - ... - 1 - 0 - 15 - 14 - ... - 9 - 8 - 7

bas centre haut

Registre 19	IL7	IL6	IL5	IL4	IL3	IL2	IL1	IL0	MSX2

IL0 à IL7 - donnent le numéro de ligne où une interruption programmée doit se produire

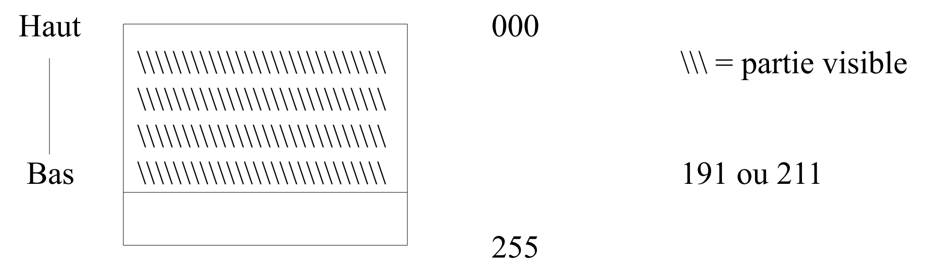
Registre 20	0	0	0	0	0	0	0	0	MSX2
Registre 21	0	0	1	1	1	0	1	1	MSX2
Registre 22	0	0	0	0	0	1	0	1	MSX2

Les valeurs de ces 3 registres sont fixes. Ne pas les modifier. Seule exception à la règle, si l'on met ces 3 registres à 0, l'affichage disparaît. Il suffit de remettre les 3 registres à leur valeur initiale pour rétablir l'affichage.

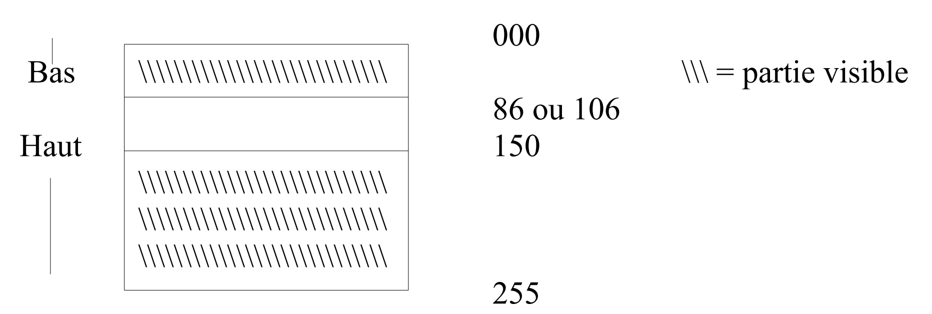
Registre 23	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0	MSX2

D0 à D7 donnent l'offset de l'affichage

Cas normal :



Exemple avec 150 dans le registre 23 :



Note : ce registre permet de réaliser très facilement des scrollings verticaux. Voici un petit programme Basic pour illustrer cette facilité :

```
10 SCREEN 8 : C = PEEK(7)+1
20 CIRCLE (100, 100), 50, 200
30 '
40 FOR I = 0 TO 50
50 OUT C, I : OUT C, &H80+23
60 NEXT
```

```
70 FOR I = 50 TO 0 STEP -1
80 OUT C, I : OUT C, &H80+23
90 NEXT
99 GOTO 40
```

L'équivalent en langage machine serait :

```
                ORG    0C000H
;
EXTROM          EQU    0015FH
CHGMOD          EQU    000D1H
CHGCLR          EQU    00062H
CLS             EQU    000C3H
NVBXLN          EQU    000C9H
BREAKX          EQU    000B7H
TOTEXT          EQU    000D2H
;
FORCLR          EQU    0F3E9H
BAKCLR          EQU    0F3EAH
BDRCLR          EQU    0F3EBH
GXPOS           EQU    0FCB3H
GYPOS           EQU    0FCB5H
ATRBYT          EQU    0F3F2H
LOGOPR          EQU    0FB02H
;
;       INITIALISATION (ECRAN, COULEURS, ETC)
;
DEBUT:          LD     A, 8
                LD     IX, CHGMOD
                CALL   EXTROM
;
                LD     A, 00AH
                LD     (FORCLR), A
                LD     A, 0
                LD     (BAKCLR), A
                LD     (BDRCLR), A
;
; TRACE D'UN RECTANGLE
                LD     BC, 060H
                LD     DE, 060H
                LD     HL, 090H
                LD     (GXPOS), HL
                LD     (GYPOS), HL
                LD     A, 0FCH
                LD     (ATRBYT), A
                LD     (LOGOPR), A
                LD     IX, NVBXLN
                CALL   EXTROM
;
; BOUCLE PRINCIPALE
;
LOOP:           CALL   UP
```

```
                CALL BREAKX
                JR      C, OUT
                CALL DOWN
                CALL  BREAKX
                JR      C, OUT
                JR      LOOP
;
OUT:            CALL  TOTEXT
                RET
;
UP:             LD      A, (7)
                LD      C, A
                INC     A
;
                LD      B, 040H
LOOP1:          LD      A, 040H
                SUB     B
                OUT     (C), A
                LD      A, 080H+017H
                OUT     (C), A
                CALL WAIT
                DJNZ  LOOP1
                RET
;
DOWN:           LD      B, 040H
LOOP2:          LD      A, 040H
                OUT     (C), A
                LD      A, 080H+017H
                OUT     (C), A
                CALL  WAIT
                DJNZ  LOOP2
                RET
;
WAIT:           PUSH  BC
                LD      BC, 002FFH
HEP:            DEC     BC
                LD      A, B
                OR      C
                JR      NZ, HEP
                POP BC
                RET
```

Les quinze derniers registres du V9938 sont utilisés lors de l'exécution des commandes. Voir le

paragraphe 5.8 « Les commandes internes du V9938 » pour plus de précisions.

Registre 32	SX7	SX6	SX5	SX4	SX3	SX2	SX1	SX0	MSX2
Registre 33	0	0	0	0	0	0	0	SX8	MSX2

SX0 à SX8 - codent l'abscisse du point d'origine (0 < X < 511)

Registre 34	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	MSX2
Registre 35	0	0	0	0	0	0	SY9	SY8	MSX2

SY0 à SY9 - codent l'ordonnée du point d'origine (0 < Y < 1023)

Registre 36	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	MSX2
Registre 37	0	0	0	0	0	0	0	DX8	MSX2

DX0 à DX8 - codent l'abscisse du point de destination (0 < X < 511)

Registre 38	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	MSX2
Registre 39	0	0	0	0	0	0	DY9	DY8	MSX2

DY0 à DY9 - codent l'ordonnée du point de destination (0 < Y < 1023)

Registre 40	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	MSX2
Registre 41	0	0	0	0	0	0	0	NX8	MSX2

NX0 à NX8 - codent la longueur (0 < NX < 511)

Registre 42	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	MSX2

Registre 43	0	0	0	0	0	0	NY9	NY8	MSX2

NY0 à NY9 - codent la hauteur (0 < NY < 1023)

Registre 44	CL7	CL6	CL5	CL4	CL3	CL2	CL1	CL0	MSX2

CL0 à CL7 - donnent la couleur

Registre 45	0	MXC	MXD	MXS	DIY	DIX	EQ	MAJ	MSX2

Registre de données pour les commandes du V9938. Nous verrons le détail des bits dans le chapitre 5.8 « Les commandes du V9938 ».

Registre 46	CM3	CM2	CM1	CM0	LO3	LO2	LO1	LO0	MSX2

CM0 à CM3 - définissent la commande du V9938 à exécuter.

LO0 à LO3 - contiennent le code de l'opérateur logique utilisé.

5.7 LES REGISTRES « READ ONLY » DU V9938

Voici la liste complète de tous les registres d'état (« status register ») accessibles en lecture à l'utilisateur et la signification des informations qu'ils contiennent.

Registre 0	F	5S	C	SN4	SN3	SN2	SN1	SN0	MSX1/MSX2

F - chaque fois que le registre est lu, ce bit passe à 0.

5S - lorsque 5 sprites (9 dans les SCREEN 5 à 8) se trouvent sur la même ligne, ce bit passe à 1.

C - quand 2 sprites entrent en collision, ce bit passe à 1.

SN0 à SN4 - donnent le numéro du 5ème sprite (ou du 9ème dans les SCREEN 5 à 8) lorsque le flag 5S se trouve à 1

Registre 1	FL	LPS	ID4	ID3	ID2	ID1	ID0	FH	MSX2

FL - flag du crayon optique. Pour que ce dernier fonctionne, il faut que le bit IE2 soit à 1, ce bit passe à 1 lorsque le crayon optique détecte de la lumière. Ce bit est automatiquement remis à 0 quand on lit le registre 1.

flag de la souris. Ce bit est à 1 si l'utilisateur a appuyé sur le bouton n°2 de la souris.

LPS - flag du bouton du crayon optique. Si le bouton a été enfoncé, ce bit passe à 1

flag de la souris. Ce bit est à 1 si l'utilisateur a appuyé sur le bouton n°1 de la souris.

ID0 à ID3 - donnent le numéro d'identification du processeur vidéo V9938

FH - flag d'interruption lors d'un balayage horizontal

Note : pour plus de renseignements sur le fonctionnement du crayon optique et de la souris, voir le paragraphe 5.11 « La souris et le crayon optique »

Registre 2	TR	VR	HR	BD	1	1	EO	CE	MSX2

TR - lorsque ce bit est à 1, le V9938 peut effectuer des transferts avec la VRAM.

VR - lors d'un balayage vertical, ce bit est à 1.

HR - lors d'un balayage horizontal, ce bit est à 1.

BD - lors d'une commande de recherche, ce bit passe à 1 lorsque la couleur de frontière est trouvée.

CE - lorsque ce bit est à 1, le V9938 est en train d'exécuter une instruction.

Registre 3	X7	X6	X5	X4	X3	X2	X1	X0	MSX2

Registre 4	1	1	1	1	1	1	1	X8	MSX2

X0 à X8 - contiennent :

- soit l'abscisse du point de collision des sprites.
- soit l'abscisse du point où pointe le crayon optique.
- soit le déplacement horizontal relatif de la souris.

Registre 5	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	MSX2

Registre 6	1	1	1	1	1	1	1	Y8	MSX2

Y0 à Y8 - contiennent :

- soit l'ordonnée du point de collision des sprites.
- soit l'ordonnée du point où pointe le crayon optique.
- soit le déplacement vertical relatif de la souris.

Registre 7	C7	C6	C5	C4	C3	C2	C1	C0	MSX2

C0 à C7 - utilisés lors des commandes « POINT » et « LMCM ».

Registre 8	BX7	BX6	BX5	BX4	BX3	BX2	BX1	BX0	MSX2

Registre 9	1	1	1	1	1	1	1	BX8	MSX2

BX0 à BX8 - renferment, lors d'une recherche (commande « SRCH » du V9938), l'abscisse du premier point trouvé.

5.8 LES COMMANDES INTERNES DU V9938

Le V9938 dispose d'un jeu d'instructions (ou commandes), tout comme le Z80. Dans cette partie, nous allons détailler les instructions une à une, et voir comment on les met en œuvre.

Le V9938 possède 12 instructions distinctes ainsi qu'une instruction d'arrêt (STOP). Voici un tableau récapitulatif :

Mnémonique	Source	Destination	Signification
HMMC	Z80	VRAM	High speed Move to Memory from CPU
YMMM	VRAM	VRAM	Y Move to Memory from Memory
HMMM	VRAM	VRAM	High speed Move to memory from memory
HMMV	V9938	VRAM	High speed Move to Memory from VDP
LMMC	Z80	VRAM	Logical Move to Memory from CPU
LMCM	VRAM	Z80	Logical Move to CPU from Memory
LMMM	VRAM	VRAM	Logical Move to Memory from memory
LMMV	V9938	VRAM	Logical Move to Memory from VDV

LINE	V9938	VRAM	LINE
SRCH	V9938	VRAM	SeaRCH
PSET	V9938	VRAM	Point SET
POINT	VRAM	V9938	is POINT set ?
STOP	-	-	STOP

Le V9938 travaille uniquement en coordonnées X et Y. Les notions d'adresse mémoire et de page n'existent pas. Tout se passe comme si le V9938 opérait sur un écran géant de 256x1024 (ou 512x1024). La partie visible ne serait alors qu'une fenêtre suivant le schéma :

SCREEN 5		VRAM	SCREEN 6	
0, 0	255, 0	00000H	0, 0	511, 0
page 0			page 0	
0, 255	255, 255	08000H	0, 255	511, 255
0, 256	255,256		0, 256	511, 256
page 1			page 1	
0, 511	255,511	10000H	0, 511	511, 511
0, 512	255,512		0, 512	511, 512
page 2			page 2	
0, 767	255, 767	18000H	0, 767	511, 767
0, 768	255, 768		0, 768	511, 768
page 3			page 3	
0, 1023	255, 1023	20000H	0, 1023	511, 1023

SCREEN 7		VRAM	SCREEN 8	
0, 0	511, 0	00000H	0, 0	255, 0
page 0		08000H	page 0	
0, 255	511, 255	10000H	0, 255	255, 255
0, 256	511, 256		0, 256	255,256
page 1		18000H	page 1	
0, 511	511, 511	20000H	0, 511	255,511

Lorsque l'on démarre une instruction, il est toujours possible de spécifier un opérateir logique grâce aux bits LO0 à LO3 sachant que :

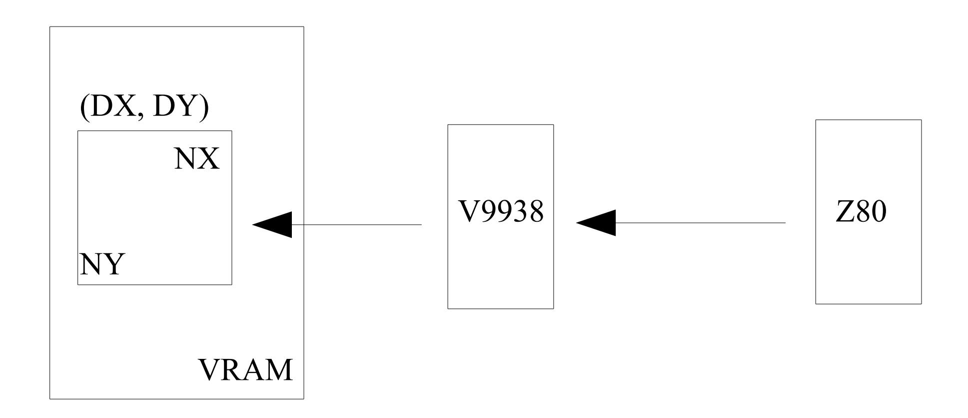
Opérateur	Code
immédiat	0000
AND	0001
OR	0010
XOR	0011
NOT	0100

Opérateur	Code
T immédiat	1000
TAND	1001
TOR	1010
TXOR	1011
TNOT	1100

Les opérateurs de la seconde colonne fonctionnent de manière identique à ceux de la première si ce n'est qu'ils n'effectuent pas d'opération logique lorsque la couleur de fond est la couleur 0.

INSTRUCTION HMMC (High speed Move to Memory from CPU)

Schéma :



Fonction : HMMC permet de remplir la mémoire vidéo avec des données provenant du Z80. A la différence des routines du Bios (BIGFIL ou FILVRM), les données servant à remplir la mémoire vidéo peuvent être toutes différentes. De plus la mémoire vidéo est définie par des coordonnées en X et Y.

Source : Z80

Destination : VRAM

Registres : Voici la liste des registres du processeur vidéo à charger pour exécuter la commande HMMC :

R36	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	abscisse
R37	0	0	0	0	0	0	0	DX8	(0 à 511)

DX0 à DX8 : abscisse du pixel origine

En SCREEN 5 et 7, le bit de poids faible est perdu.

En SCREEN 6, les bits DX0 et DX1 sont perdus.

R38	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	ordonnée
R39	0	0	0	0	0	0	DY9	DY8	(0 à 1023)

DY0 à DY9 : ordonnée du pixel d'origine

R40	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	déplacement horizontal
R41	0	0	0	0	0	0	0	NX8	(0 à 511)

NX0 à NX8 : nombre de pixels à allumer dans la direction horizontale.

En SCREEN 5 et 7, le bit NX0 est perdu.

En SCREEN 6, les 2 bits NX0 et NX1 sont perdus.

R42	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	déplacement vertical
R43	0	0	0	0	0	0	NY9	NY8	(0 à 1023)

NY0 à NY9 - nombre de pixels à allumer dans la direction verticale.

R44	CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0	donnée

SCREEN 5 et 7 : les bits CR4 à CR7 codent la couleur du pixel d'abscisse pair alors que CR0 à CR3 donnent la couleur du pixel d'abscisse impair.

SCREEN 6 : CR6 et CR7 codent la couleur du premier pixel (dont l'abscisse doit toujours être multiple de 4). CR4 et CR5 donnent la couleur du pixel immédiatement à droite du pixel précédent et ainsi de suite pour les bits C2 et C3 d'une part, C0 et C1 d'autre part.

SCREEN 8 : CR0 à CR7 codent la couleur d'un pixel.

R45	0	0	MXD	0	DIY	DIX	0	0	paramètres

MXD : 0 pour sélectionner la VRAM principale.

1 pour accéder la mémoire vidéo étendue sur les MSX2 avec 192 Ko de VRAM.

DIY : direction verticale pour NY : 0 = bas, 1 = haut.

DIX : direction horizontale pour NX : 0 = droite, 1 = gauche.

Exécution : pour exécuter l'instruction HMMC, voici la marche à suivre :

1 - charger les registres ci-dessus

2 - mettre à 11110000B (0F0H) le registre 46.

3 - envoyer l'octet suivant à mettre en VRAM dans le registre 45 (le premier octet a été traité à l'étape 1) par un OUT du Z80.

4 - lire le registre d'état 2 (status).

5 - lire le registre d'état du bit CE, si celui-ci est à 0, alors l'instruction est terminée, sinon on passe à l'étape 6.

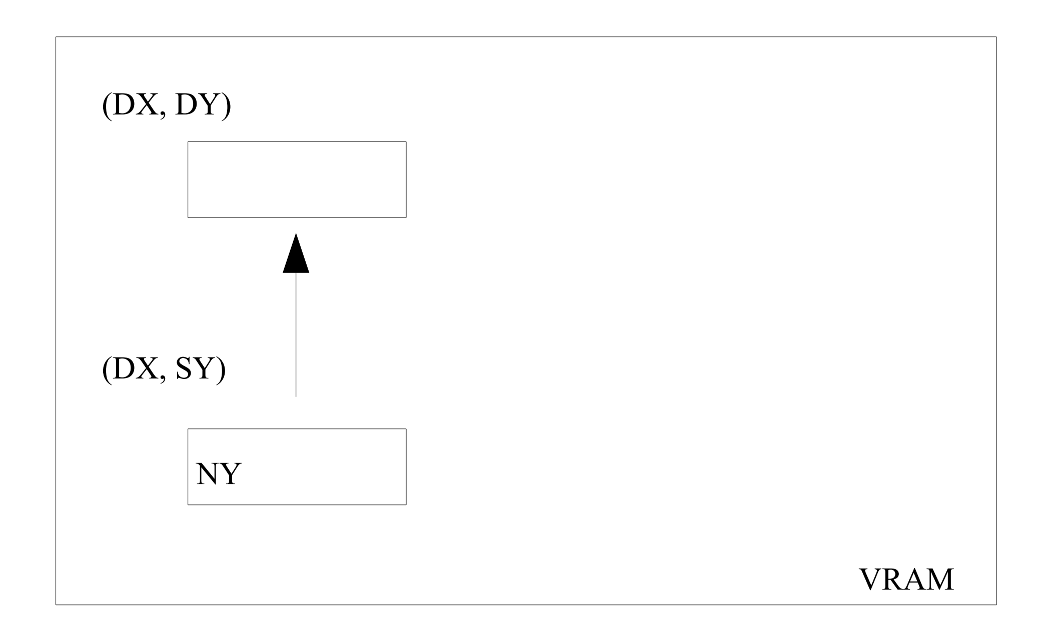
6 - tester l'état du bit TR, si celui-ci se trouve à 0, alors le processeur vidéo n'est pas prêt à recevoir l'octet suivant, recommencer en 4. Si par contre, ce bit est à 1, reprendre toute l'opération au niveau 3.

Etat : Une fois l'instruction terminée, les registres su processeur vidéo se trouvent dans l'état suivant :

32-33	34-35	36-37	38-39	40-41	42-43	44	45
		idem	modif	idem	modif	idem	idem

INSTRUCTION YMMM (Y Move to Memory from Memory)

Schéma :



Fonction : YMMM autorise la copie d'octets d'une zone de mémoire vidéo à une autre, le point d'origine de cette dernière ayant la même abscisse.

Source : VRAM

Destination : VRAM

Registres : Voici la liste des registres du processeur vidéo à charger pour exécuter la commande :

R34	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	ordonnée source
R35	0	0	0	0	0	0	SY9	SY8	(0 à 1023)

SY0 à SY9 : ordonnée du pixel origine.

R36	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	abscisse destination
R37	0	0	0	0	0	0	0	DX8	(0 à 511)

DX0 à DX8 : abscisse du pixel destination

En SCREEN 5 et 7, le bit de poids faible est perdu.

En SCREEN 6, les bits DX0 et DX1 sont perdus.

R38	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	ordonnée destination
R39	0	0	0	0	0	0	DY9	DY8	(0 à 1023)

DY0 à DY9 : ordonnée du pixel destination

R42	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	déplacement vertical
R43	0	0	0	0	0	0	NY9	NY8	(0 à 1023)

NY0 à NY9 - nombre de pixels à copier dans la direction verticale.

R45	0	0	MXD	0	DIY	DIX	0	0	paramètres

MXD : 0 pour sélectionner la VRAM principale.

1 pour accéder à la mémoire vidéo étendue sur les MSX2 avec 192 Ko de VRAM.

DIY : direction verticale pour NY : 0 = bas, 1 = haut.

DIX : direction horizontale pour NX : 0 = droite, 1 = gauche.

Exécution : pour exécuter l'instruction YMMM, voici la marche à suivre :

1 - charger les registres ci-dessus.

2 - mettre 11100000B (0E0H) dans le registre 46.

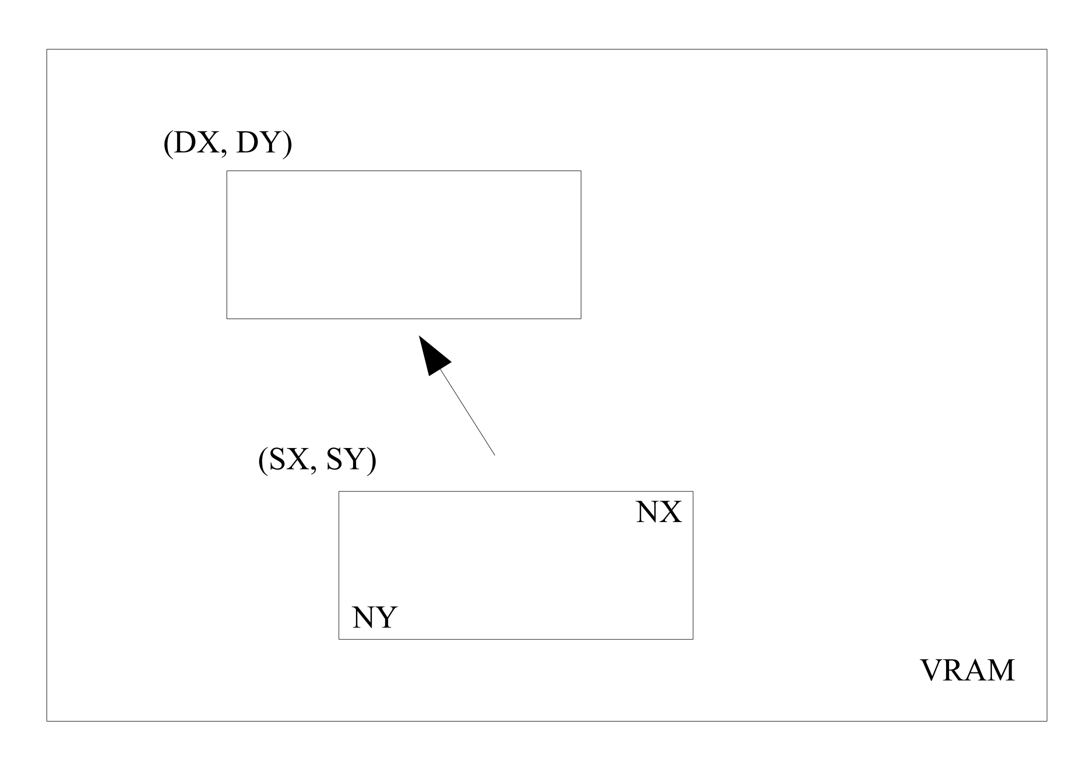
3 - avant un autre accès au processeur vidéo, lire le registre d'état 2 (status), tester l'état du bit CE. Si celui-ci est à 0, alors l'instruction est terminée, le V9938 est disponible, sinon (bit CE à 1), il faut attendre (ou faire autre chose) avant d'utiliser à nouveau le processeur vidéo.

Etat : Une fois l'instruction terminée, les registres su processeur vidéo se trouvent dans l'état suivant :

32-33	34-35	36-37	38-39	40-41	42-43	44	45
	modif	idem	modif		modif		idem

INSTRUCTION HMMM (High Speed Move to Memory from Memory)

Schéma :



Fonction : HMMM autorise la copie rapide d'octets d'une zone de mémoire vidéo à une autre de taille équivalente.

Source : VRAM

Destination : VRAM

Registres : Voici la liste des registres du processeur vidéo à charger pour exécuter la commande :

R32	SX7	SX6	SX5	SX4	SX3	SX2	SX1	SX0	abscisse source
R33	0	0	0	0	0	0	0	SX8	(0 à 511)

SX0 à SX8 : abscisse du pixel origine

R34	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	ordonnée source
R35	0	0	0	0	0	0	SY9	SY8	(0 à 1023)

SY0 à SY9 : ordonnée du pixel origine.

R36	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	abscisse destination
R37	0	0	0	0	0	0	0	DX8	(0 à 511)

DX0 à DX8 : abscisse du pixel destination
En SCREEN 5 et 7, le bit de poids faible est perdu.
En SCREEN 6, les bits DX0 et DX1 sont perdus.

R38	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	ordonnée destination
R39	0	0	0	0	0	0	DY9	DY8	(0 à 1023)

DY0 à DY9 : ordonnée du pixel destination

R40	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	déplacement horizontal
R41	0	0	0	0	0	0	0	NX8	(0 à 511)

NX0 à NX8 : nombre de pixels à copier dans la direction horizontale

R42	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	déplacement vertical
R43	0	0	0	0	0	0	NY9	NY8	(0 à 1023)

NY0 à NY9 : nombre de pixels à copier dans la direction verticale.

R45	0	0	MXD	MXS	DIY	DIX	0	0	paramètres

MXD : Mémoire vidéo de destination
0 pour sélectionner la VRAM principale.
1 pour accéder à la mémoire vidéo étendue sur les MSX2 avec 192 Ko de VRAM.

MXS : Mémoire vidéo source
0 pour sélectionner la VRAM principale.
1 pour accéder à la mémoire vidéo étendue sur les MSX2 avec 192 Ko de VRAM.

DIY : direction verticale pour NY : 0 = bas, 1 = haut.

DIX : direction horizontale pour NX : 0 = droite, 1 = gauche.

Exécution : pour exécuter l'instruction HMMM, voici la marche à suivre :
1 - charger les registres ci-dessus
2 - mettre à 11010000B (0D0H) le registre 46
3 - avant un autre accès au processeur vidéo, lire le registre d'état 2 (status), tester

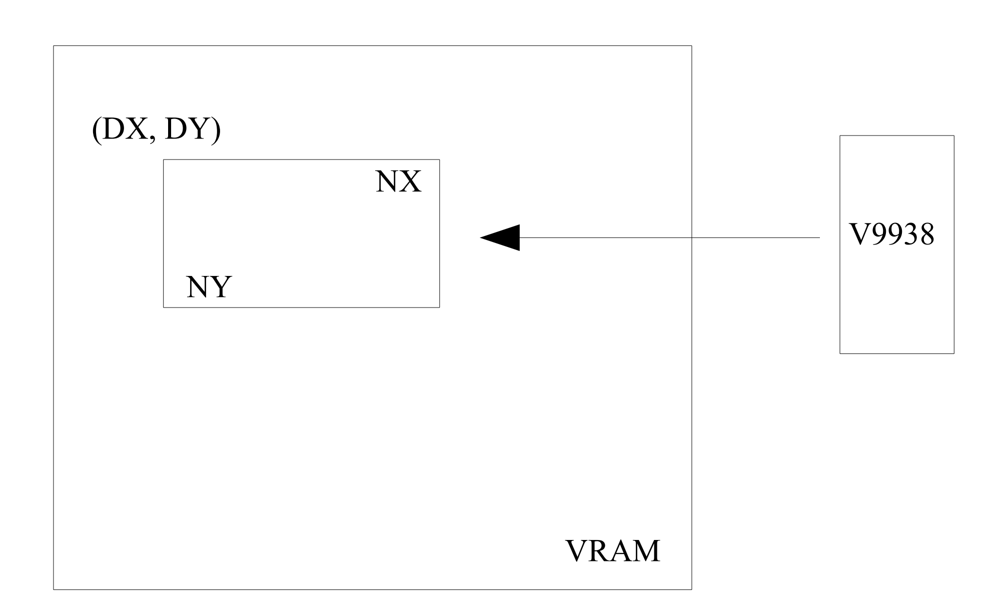
l'état du bit CE. Si celui-ci est à 0, alors l'instruction est terminée, le V9938 est disponible, sinon (bit CE à 1), il faut attendre (ou faire autre chose) avant d'utiliser à nouveau le processeur vidéo.

Etat : Une fois l'instruction terminée, les registres su processeur vidéo se trouvent dans l'état suivant :

32-33	34-35	36-37	38-39	40-41	42-43	44	45
idem	modif	idem	modif	idem	modif		idem

INSTRUCTION HMMV (High speed Move to Memory from VDP)

Schéma :



Fonction : HMMV permet de remplir la mémoire vidéo avec une seule donnée. A la différence des routines du Bios (BIGFIL ou FILVRM), la mémoire vidéo est définie par des coordonnées en X et Y.

Source : V9938

Destination : VRAM

Registres : Voici la liste des registres du processeur vidéo à charger pour exécuter la commande :

R36	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	abscisse
R37	0	0	0	0	0	0	0	DX8	(0 à 511)

DX0 à DX8 : abscisse du pixel origine

En SCREEN 5 et 7, le bit de poids faible est perdu.

En SCREEN 6, les bits DX0 et DX1 sont perdus.

R38	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	ordonnée
R39	0	0	0	0	0	0	DY9	DY8	(0 à 1023)

DY0 à DY9 : ordonnée du pixel origine

R40	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	déplacement horizontal
R41	0	0	0	0	0	0	0	NX8	(0 à 511)

NX0 à NX8 : nombre de pixels à allumer dans la direction horizontale

En SCREEN 5 et 7, le bit NX0 est perdu.

En SCREEN 6, les bits NX0 et NX1 sont perdus.

R42	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	déplacement vertical
R43	0	0	0	0	0	0	NY9	NY8	(0 à 1023)

NY0 à NY9 : nombre de pixels à allumer dans la direction verticale.

R44	CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0	donnée

SCREEN 5 et 7 : les bits CR4 à CR7 codent la couleur du pixel d'abscisse pair alors que CR0 à CR3 donnent la couleur du pixel d'abscisse impair.

SCREEN 6 : CR6 et CR7 codent la couleur du premier pixel (dont l'abscisse doit toujours être multiple de 4). CR4 et CR5 donnent la couleur du pixel immédiatement à droite du pixel précédent et ainsi de suite pour les bits C2 et C3 d'une part, C0 et C1 d'autre part.

SCREEN 8 : CR0 à CR7 codent la couleur d'un pixel.

Dans tous les modes, il faut charger le registre 44 avec l'octet à écrire en VRAM.

R45	0	0	MXD	0	DIY	DIX	0	0	paramètres

MXD : 0 pour sélectionner la VRAM principale.

1 pour accéder à la mémoire vidéo étendue sur les MSX2 avec 192 Ko de VRAM.

DIY : direction verticale pour NY : 0 = bas, 1 = haut.

DIX : direction horizontale pour NX : 0 = droite, 1 = gauche.

Exécution : pour exécuter l'instruction HMMV, voici la marche à suivre :

1 - charger les registres ci-dessus

2 - mettre à 11000000B (0C0H) le registre 46

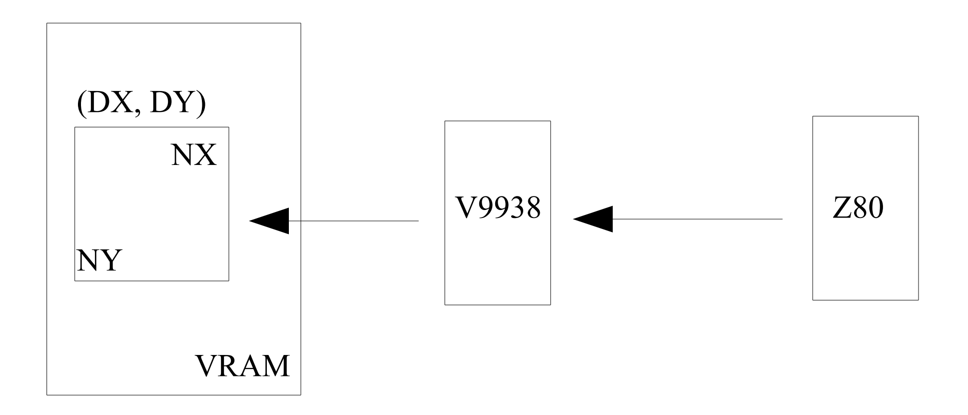
3 - avant un autre accès au processeur vidéo, lire le registre d'état 2 (status), tester l'état du bit CE. Si celui-ci est à 0, alors l'instruction est terminée, le V9938 est disponible, sinon (bit CE à 1), il faut attendre (ou faire autre chose) avant d'utiliser à nouveau le processeur vidéo.

Etat : Une fois l'instruction terminée, les registres su processeur vidéo se trouvent dans l'état suivant :

32-33	34-35	36-37	38-39	40-41	42-43	44	45
		idem	modif	idem	modif	idem	idem

INSTRUCTION LMMC (Logical Move to Memory from CPU)

Schéma :



Fonction : LMMC permet de remplir la mémoire vidéo avec des données provenant du Z80. Le remplissage se fait pixel par pixel. Cette instruction s'exécute donc moins rapidement que HMMC. En contrepartie, il est possible de définir un opérateur logique. A la différence des routines du Bios (BIGFIL ou FILVRM), les données servant à remplir la mémoire vidéo peuvent être toutes différentes. De plus, la mémoire vidéo est définie par des coordonnées X e Y.

Source : Z80

Destination : VRAM

Registres : Voici la liste des registres du processeur vidéo à charger pour exécuter la commande :

R36	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	abscisse
R37	0	0	0	0	0	0	0	DX8	(0 à 511)

DX0 à DX8 : abscisse du pixel origine

R38	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	ordonnée
R39	0	0	0	0	0	0	DY9	DY8	(0 à 1023)

DY0 à DY9 : ordonnée du pixel origine

R40	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	déplacement horizontal
R41	0	0	0	0	0	0	0	NX8	(0 à 511)

NX0 à NX8 : nombre de pixels à allumer dans la direction horizontale

R42	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	déplacement vertical
R43	0	0	0	0	0	0	NY9	NY8	(0 à 1023)

NY0 à NY9 : nombre de pixels à allumer dans la direction verticale.

R44	CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0	donnée

SCREEN 5 et 7 : seuls les bits CR0 à CR3 sont utilisés. Ils codent la couleur d'un pixel.

SCREEN 6 : CR0 et CR1 codent la couleur d'un pixel. Les autres bits ne soont pas utilisés.

SCREEN 8 : CR0 à CR7 codent la couleur d'un pixel.

R45	0	0	MXD	0	DIY	DIX	0	0	paramètres

MXD : 0 pour sélectionner la VRAM principale.

1 pour accéder à la mémoire vidéo étendue sur les MSX2 avec 192 Ko de VRAM.

DIY : direction verticale pour NY : 0 = bas, 1 = haut.

DIX : direction horizontale pour NX : 0 = droite, 1 = gauche.

R46	1	0	1	1	LO3	LO2	LO1	LO0	paramètres

LO0 à LO3 : opérateur logique à appliquer

0000 = direct	1000 = Tdirect
0001 = AND	1001 = TAND
0010 = OR	1010 = TOR
0011 = XOR	1011 = TXOR
0100 = NOT	1100 = TNOT

Exécution : pour exécuter l'instruction LMMC, voici la marche à suivre :

1 - charger les registres ci-dessus sauf le 46

2 - mettre à 1011B les 4 bits de poids fort du registre 46, charger les 4 bits de poids faible avec le code de l'opérateur logique désiré.

3 - envoyer l'octet suivant à mettre en VRAM dans le registre 45 (le premier octet a été traité à l'étape 1) par un OUT du Z80.

4 - lire le registre d'état 2 (status),

5 - tester l'état du bit CE. Si celui-ci est à 0, alors l'instruction est terminée, sinon on passe à l'étape 6.

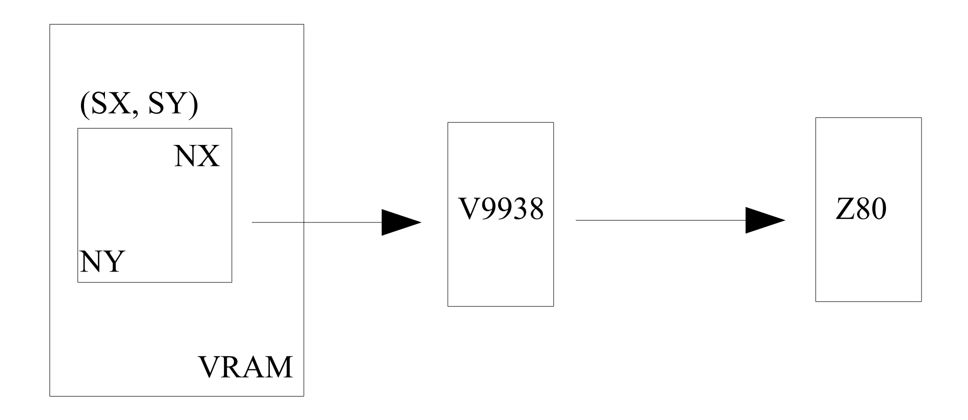
6 - tester l'état du bit TR, si celui-ci se trouve à 0, alors le processeur vidéo n'est pas prêt à recevoir l'octet suivant, recommencer en 4. Si par contre ce bit est à 1, reprendre toute l'opération au niveau 3.

Etat : Une fois l'instruction terminée, les registres su processeur vidéo se trouvent dans l'état suivant :

32-33	34-35	36-37	38-39	40-41	42-43	44	45
		idem	modif	idem	modif	idem	idem

INSTRUCTION LMCM (Logical Move to CPU from Memory)

Schéma :



Fonction : LMCM permet de récupérer en mémoire central (via le Z80) le contenu d'une zone de la mémoire vidéo. Le transfert de fait pixel par pixel.

Source : VRAM

Destination : Z80

Registres : Voici la liste des registres du processeur vidéo à charger pour exécuter la commande :

R32	SX7	SX6	SX5	SX4	SX3	SX2	SX1	SX0	abscisse
R33	0	0	0	0	0	0	0	SX8	(0 à 511)

SX0 à SX8 : abscisse du pixel origine

R34	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	ordonnée
R35	0	0	0	0	0	0	SY9	SY8	(0 à 1023)

SY0 à SY9 : ordonnée du pixel origine.

R40	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	déplacement horizontal
R41	0	0	0	0	0	0	0	NX8	(0 à 511)

NX0 à NX8 : nombre de pixels à transférer dans la direction horizontale

R42	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	déplacement vertical
R43	0	0	0	0	0	0	NY9	NY8	(0 à 1023)

NY0 à NY9 : nombre de pixels à transférer dans la direction verticale.

R45	0	0	0	MXS	DIY	DIX	0	0	paramètres

MXS : 0 pour sélectionner la VRAM principale.

1 pour accéder à la mémoire vidéo étendue sur les MSX2 avec 192 Ko de VRAM.

DIY : direction verticale pour NY : 0 = bas, 1 = haut.

DIX : direction horizontale pour NX : 0 = droite, 1 = gauche.

Exécution : pour exécuter l'instruction HMMM, voici la marche à suivre :

1 - charger les registres ci-dessus

2 - mettre à 10100000B (0A0H) le registre 46

3 - lire le registre d'état 2 (status)

4 - tester l'état du bit TR, si celui-ci se trouve à 0, alors le processeur vidéo n'est pas prêt à recevoir l'octet suivant, recommencer en 3. Si par contre, ce bit est à 1, lire le registre d'état 7 (status). Il contient la donnée issue de la mémoire vidéo.

5 - tester l'état du bit CE, si celui-ci est à 0, alors l'instruction est terminée, il suffit de lire le dernier octet dans le registre d'état 7. Dans le cas contraire (CE à 1), on recommence à l'étape 3.

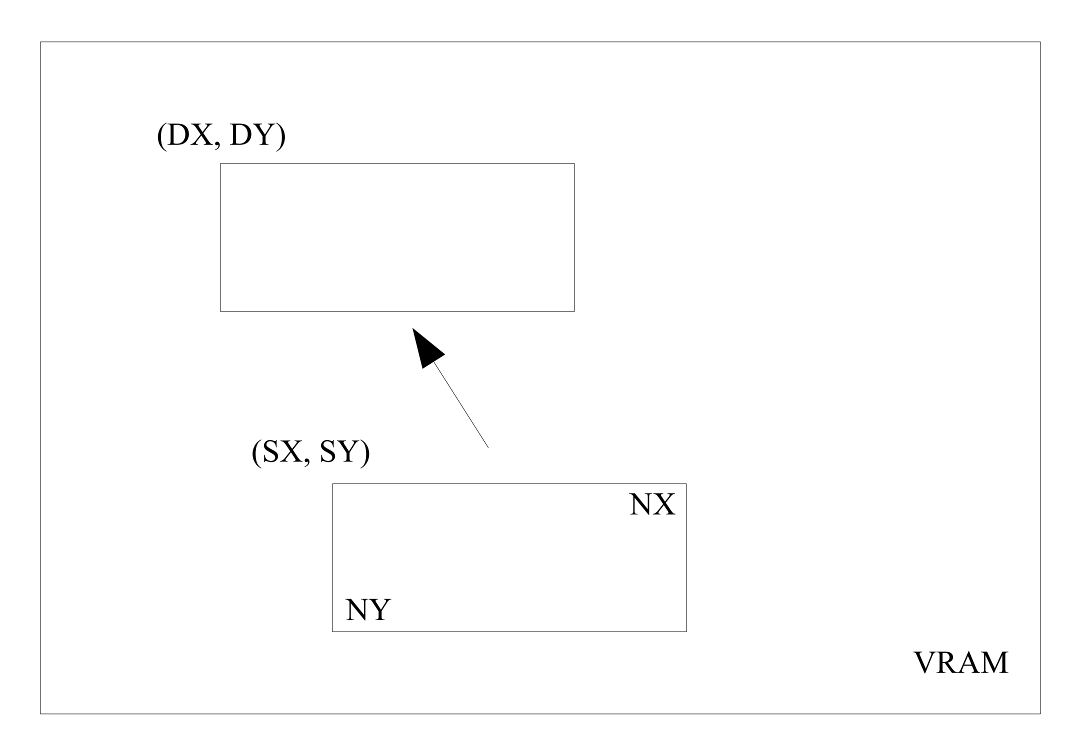
ATTENTION : le bit TR du registre d'état 2 doit toujours être à 0 avant de commencer l'exécution de cette instruction (ceci peut s'obtenir facilement en lisant l'état du registre d'état 7).

Etat : Une fois l'instruction terminée, les registres su processeur vidéo se trouvent dans l'état suivant :

32-33	34-35	36-37	38-39	40-41	42-43	44	45
idem	modif			idem	modif	code	idem

INSTRUCTION LMMM (Logical Move to Memory from Memory)

Schéma :



Fonction : LMMM autorise la copie d'une zone de mémoire vidéo vers une autre de taille équivalente. La copie se fait pixel par pixel

Source : VRAM

Destination : VRAM

Registres : Voici la liste des registres du processeur vidéo à charger pour exécuter la commande :

R32	SX7	SX6	SX5	SX4	SX3	SX2	SX1	SX0	abscisse source
R33	0	0	0	0	0	0	0	SX8	(0 à 511)

SX0 à SX8 : abscisse du pixel origine

R34	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	ordonnée source
R35	0	0	0	0	0	0	SY9	SY8	(0 à 1023)

SY0 à SY9 : ordonnée du pixel origine.

R36	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	abscisse destination
R37	0	0	0	0	0	0	0	DX8	(0 à 511)

DX0 à DX8 : abscisse du pixel destination

R38	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	ordonnée destination
R39	0	0	0	0	0	0	DY9	DY8	(0 à 1023)

DY0 à DY9 : ordonnée du pixel destination

R40	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	déplacement horizontal
R41	0	0	0	0	0	0	0	NX8	(0 à 511)

NX0 à NX8 : nombre de pixels à copier dans la direction horizontale

R42	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	déplacement vertical
R43	0	0	0	0	0	0	NY9	NY8	(0 à 1023)

NY0 à NY9 : nombre de pixels à copier dans la direction verticale.

R45	0	0	MXD	MXS	DIY	DIX	0	0	paramètres

MXD : Mémoire vidéo de destination
0 pour sélectionner la VRAM principale.
1 pour accéder à la mémoire vidéo étendue sur les MSX2 avec 192 Ko de VRAM.

MXS : Mémoire vidéo source
0 pour sélectionner la VRAM principale.
1 pour accéder à la mémoire vidéo étendue sur les MSX2 avec 192 Ko de VRAM.

DIY : direction verticale pour NY : 0 = bas, 1 = haut.

DIX : direction horizontale pour NX : 0 = droite, 1 = gauche.

R46	1	0	0	1	LO3	LO2	LO1	LO0	commandes

LO0 à LO3 : opérateur logique à appliquer

0000 = direct   1000 = Tdirect
0001 = AND   1001 = TAND
0010 = OR   1010 = TOR
0011 = XOR   1011 = TXOR
0100 = NOT   1100 = TNOT

Exécution : pour exécuter l'instruction LMMM, voici la marche à suivre :
1 - charger les registres ci-dessus sauf le 46

2 - mettre à 1001B (09H) les 4 bits de poids fort du registre 46, charger les 4 bits de poids faible avec le code de l'opérateur logique désiré.

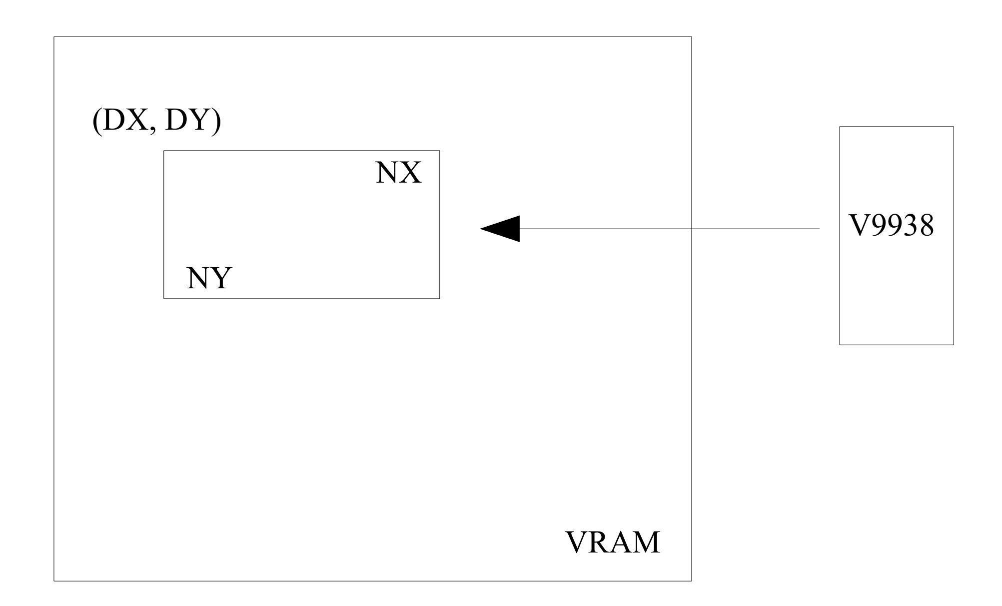
3 - avant un autre accès au processeur vidéo, lire le registre d'état 2 (status), tester l'état du bit CE. Si celui-ci est à 0, alors l'instruction est terminée, le V9938 est disponible, sinon (bit CE à 1), il faut attendre (ou faire autre chose) avant d'utiliser à nouveau le processeur vidéo.

Etat : Une fois l'instruction terminée, les registres su processeur vidéo se trouvent dans l'état suivant :

32-33	34-35	36-37	38-39	40-41	42-43	44	45
idem	modif	idem	modif	idem	modif		idem

INSTRUCTION LMMV (Logical Move to Memory from VDP)

Schéma :



Fonction : LMMV permet de remplir la mémoire vidéo avec une seule donnée. A la différence des routines du Bios (BIGFIL ou FILVRM), la mémoire vidéo est définie par des coordonnées en X et Y. Le remplissage se fait pixel par pixel, ce qui autorise l'emploi d'un opérateur logique.

Source : V9938

Destination : VRAM

Registres : Voici la liste des registres du processeur vidéo à charger pour exécuter la commande :

R36	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	abscisse
R37	0	0	0	0	0	0	0	DX8	(0 à 511)

DX0 à DX8 : abscisse du pixel origine

R38	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	ordonnée
R39	0	0	0	0	0	0	DY9	DY8	(0 à 1023)

DY0 à DY9 : ordonnée du pixel origine

R40	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	déplacement horizontal
R41	0	0	0	0	0	0	0	NX8	(0 à 511)

NX0 à NX8 : nombre de pixels à allumer dans la direction horizontale

R42	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	déplacement vertical
R43	0	0	0	0	0	0	NY9	NY8	(0 à 1023)

NY0 à NY9 : nombre de pixels à allumer dans la direction verticale.

R44	CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0	donnée

SCREEN 5 et 7 : seuls les bits CR0 à CR3 sont utilisés. Ils codent la couleur d'un pixel.

SCREEN 6 : CR0 et CR1 codent la couleur d'un pixel. Les autres bits ne soont pas utilisés.

SCREEN 8 : CR0 à CR7 codent la couleur d'un pixel.

R45	0	0	MXD	0	DIY	DIX	0	0	paramètres

MXD : 0 pour sélectionner la VRAM principale.

1 pour accéder à la mémoire vidéo étendue sur les MSX2 avec 192 Ko de VRAM.

DIY : direction verticale pour NY : 0 = bas, 1 = haut.

DIX : direction horizontale pour NX : 0 = droite, 1 = gauche.

R46	1	0	0	0	LO3	LO2	LO1	LO0	commandes

LO0 à LO3 : opérateur logique à appliquer

0000 = direct	1000 = Tdirect
0001 = AND	1001 = TAND
0010 = OR	1010 = TOR
0011 = XOR	1011 = TXOR
0100 = NOT	1100 = TNOT

Exécution : pour exécuter l'instruction LMMV, voici la marche à suivre :

1 - charger les registres ci-dessus sauf le 46

2 - mettre à 1000B (08H) les 4 bits de poids fort du registre 46, charger les 4 bits de poids faible avec le code de l'opérateur logique désiré.

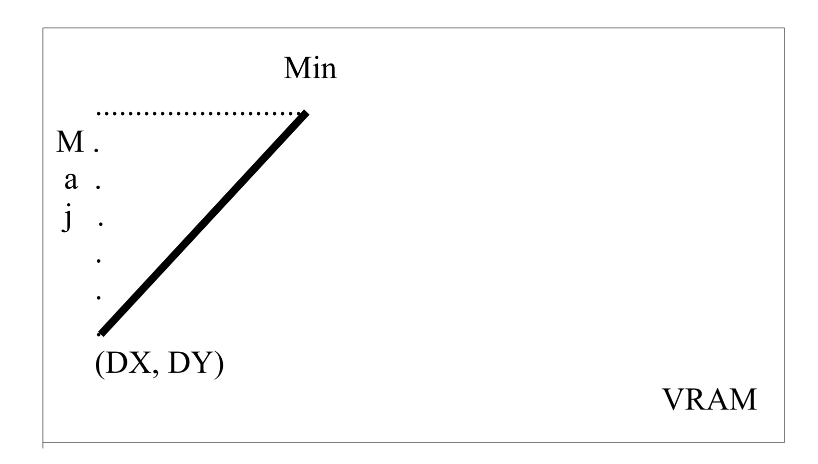
3 - avant un autre accès au processeur vidéo, lire le registre d'état 2 (status), tester l'état du bit CE. Si celui-ci est à 0, alors l'instruction est terminée, le V9938 est disponible, sinon (bit CE à 1), il faut attendre (ou faire autre chose) avant d'utiliser à nouveau le processeur vidéo.

Etat : Une fois l'instruction terminée, les registres su processeur vidéo se trouvent dans l'état suivant :

32-33	34-35	36-37	38-39	40-41	42-43	44	45
		idem	modif	idem	modif	idem	idem

INSTRUCTION LINE (draw a LINE)

Schéma :



Fonction : LINE trace une ligne dans la mémoire vidéo. En réalité, l'utilisateur définit un triangle rectangle et le processeur vidéo en trace l'hypoténuse.

Source : V9938

Destination : VRAM

Registres : Voici la liste des registres du processeur vidéo à charger pour exécuter la commande :

R36	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	abscisse destination
R37	0	0	0	0	0	0	0	DX8	(0 à 511)

DX0 à DX8 : abscisse du pixel origine.

R38	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	ordonnée destination
R39	0	0	0	0	0	0	DY9	DY8	(0 à 1023)

DY0 à DY9 : ordonnée du pixel origine.

R40	MJ7	MJ6	MJ5	MJ4	MJ3	MJ2	MJ1	MJ0	nb pixels grand côté
R41	0	0	0	0	0	0	MJ9	MJ8	(0 à 1023)

NX0 à NX8 : nombre de pixels sur le plus long des côtés.

R42	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0	nb pixels petit côté
R43	0	0	0	0	0	0	MI9	MI8	(0 à 511)

NY0 à NY9 : nombre de pixels sur le moins long des côtés.

R44	CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0	donnée

SCREEN 5 et 7 : seuls les bits CR0 à CR3 sont utilisés. Ils codent la couleur de la ligne (0-15)

SCREEN 6 : CR0 et CR1 codent la couleur de la ligne. Les autres bits (CR2 à CR7) ne sont pas utilisés.

SCREEN 8 : CR0 à CR7 codent la couleur de la ligne.

R45	0	0	MXD	0	DIY	DIX	0	MAJ	paramètres

MXD : Mémoire vidéo de destination

0 pour sélectionner la VRAM principale.

1 pour accéder à la mémoire vidéo étendue sur les MSX2 avec 192 Ko de VRAM.

DIY : direction verticale pour le tracé à partir du pixel origine :

0 = la ligne partira vers le bas

1 = la ligne partira vers le haut

DIX : direction horizontale pour le tracé à partir du pixel origine :

0 = la ligne partira vers la droite

1 = la ligne partira vers la gauche

MAJ : côté le plus long :

0 = côté horizontal

1 = côté vertical

R46	0	1	1	1	LO3	LO2	LO1	LO0	commandes

LO0 à LO3 : opérateur logique à appliquer

0000 = direct	1000 = Tdirect
0001 = AND	1001 = TAND
0010 = OR	1010 = TOR
0011 = XOR	1011 = TXOR
0100 = NOT	1100 = TNOT

Exécution : pour exécuter l'instruction LINE, voici la marche à suivre :

1 - charger les registres ci-dessus sauf le 46

2 - mettre à 0111B (07H) les 4 bits de poids fort du registre 46, charger les 4 bits de poids faible avec le code de l'opérateur logique désiré.

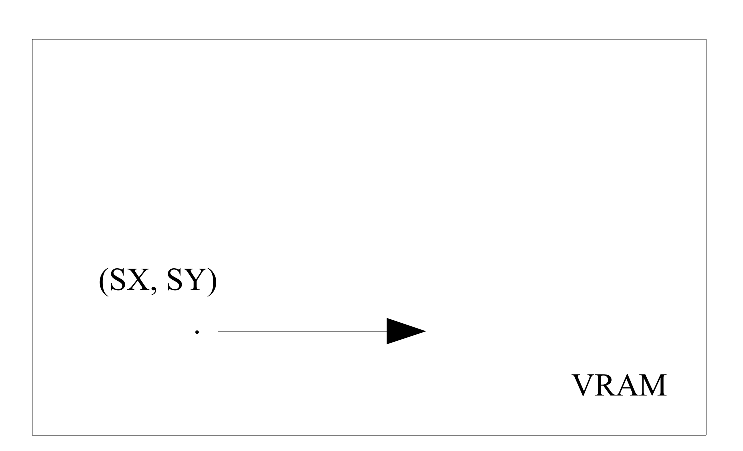
3 - avant un autre accès au processeur vidéo, lire le registre d'état 2 (status), tester l'état du bit CE. Si celui-ci est à 0, alors l'instruction est terminée, le V9938 est disponible, sinon (bit CE à 1), il faut attendre (ou faire autre chose) avant d'utiliser à nouveau le processeur vidéo.

Etat : Une fois l'instruction terminée, les registres su processeur vidéo se trouvent dans l'état suivant :

32-33	34-35	36-37	38-39	40-41	42-43	44	45
		idem	modif	idem	idem	idem	idem

INSTRUCTION SRCH (SeaRCH for border color)

Schéma :



Fonction : SRCH permet de rechercher une couleur donnée sur une ligne horizontale. Cette instruction est particulièrement utile lors du remplissage d'une surface (PAINT).

Source : V9938

Destination : VRAM

Registres : Voici la liste des registres du processeur vidéo à charger pour exécuter la commande :

R32	SX7	SX6	SX5	SX4	SX3	SX2	SX1	SX0	abscisse
R33	0	0	0	0	0	0	0	SX8	(0 à 511)

SX0 à SX8 : abscisse du pixel origine

R34	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	ordonnée
R35	0	0	0	0	0	0	SY9	SY8	(0 à 1023)

SY0 à SY9 : ordonnée du pixel origine.

R44	CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0	donnée

SCREEN 5 et 7 : seuls les bits CR0 à CR3 sont utilisés. Ils codent la couleur à rechercher (0-15)

SCREEN 6 : CR0 et CR1 codent la couleur à rechercher. Les autres bits (CR2 à CR7) ne sont pas utilisés.

SCREEN 8 : CR0 à CR7 codent la couleur à rechercher (0-255).

R45	0	0	MXD	0	0	DIX	EQ	0	paramètres

MXD : 0 pour sélectionner la VRAM principale.

1 pour accéder à la mémoire vidéo étendue sur les MSX2 avec 192 Ko de VRAM.

DIX : sens de la recherche :

0 = droite

1 = gauche

Exécution : pour exécuter l'instruction SRCH, voici la marche à suivre :

1 - charger les registres ci-dessus

2 - mettre à 01100000B (060H) le registre 46

3 - lire le registre d'état 2 (status)

4 - tester l'état du bit CE, si celui-ci est à 1, il faut patienter, on retourne à l'étape 3. Si le bit CE est à 0, on continue.

5 - tester l'état du bit BD si celui-ci se trouve à 0, alors le processeur vidéo n'a pas trouvé de pixel de la couleur recherchée. Si, au contraire, BD est à 1, le processeur vidéo a bien repéré un pixel de la couleur cherchée. L'abscisse de ce pixel s'obtient en lisant le contenu des registres d'état 8 et 9. Le premier renferme les 8 bits de poids faible de cette abscisse, alors que le bit 0 du registre 9 donne le bit de poids fort.

Etat : Une fois l'instruction terminée, les registres su processeur vidéo se trouvent dans l'état suivant :

32-33	34-35	36-37	38-39	40-41	42-43	44	45
idem	idem					idem	idem

INSTRUCTION PSET (Point SET)

Schéma :

.
(DX, DY)

VRAM

Fonction : PSET affiche un point dans la mémoire vidéo

Source : V9938

Destination : VRAM

Registres : Voici la liste des registres du processeur vidéo à charger pour exécuter la commande :

R36	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	abscisse
R37	0	0	0	0	0	0	0	DX8	(0 à 511)

DX0 à DX8 : abscisse du pixel origine.

R38	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	ordonnée
R39	0	0	0	0	0	0	DY9	DY8	(0 à 1023)

DY0 à DY9 : ordonnée du pixel origine.

R44	CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0	couleur

SCREEN 5 et 7 : seuls les bits CR0 à CR3 sont utilisés. Ils codent la couleur du pixel (0-15)

SCREEN 6 : CR0 et CR1 codent la couleur du pixel. Les autres bits (CR2 à CR7) ne sont pas utilisés.

SCREEN 8 : CR0 à CR7 codent la couleur du pixel (0-255).

R45	0	0	MXD	0	0	0	0	0	paramètres

MXD : 0 pour sélectionner la VRAM principale.

1 pour accéder à la mémoire vidéo étendue sur les MSX2 avec 192 Ko de VRAM.

R46	0	1	0	1	LO3	LO2	LO1	LO0	commandes

LO0 à LO3 : opérateur logique à appliquer

0000 = direct 1000 = Tdirect

0001 = AND 1001 = TAND

0010 = OR 1010 = TOR

0011 = XOR 1011 = TXOR

0100 = NOT 1100 = TNOT

Exécution : pour exécuter l'instruction PSET, voici la marche à suivre :

1 - charger les registres ci-dessus sauf le 46

2 - mettre à 0101B (05H) les 4 bits de poids fort du registre 46, charger les 4 bits de poids faible avec le code de l'opérateur logique désiré.

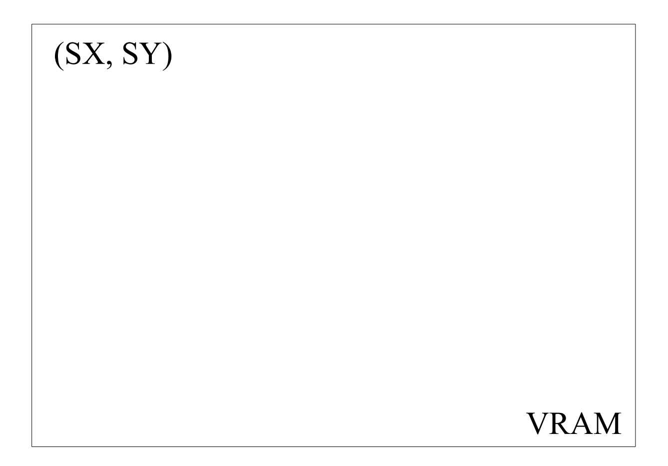
3 - avant un autre accès au processeur vidéo, lire le registre d'état 2 (status), tester l'état du bit CE. Si celui-ci est à 0, alors l'instruction est terminée, le V9938 est disponible, sinon (bit CE à 1), il faut attendre (ou faire autre chose) avant d'utiliser à nouveau le processeur vidéo.

Etat : Une fois l'instruction terminée, les registres su processeur vidéo se trouvent dans l'état suivant :

32-33	34-35	36-37	38-39	40-41	42-43	44	45
		idem	idem			idem	idem

INSTRUCTION POINT (is POINT set ?)

Schéma :



Fonction : POINT donne la couleur d'un pixel dans la mémoire vidéo.

Source : VRAM

Destination : V9938

Registres : Voici la liste des registres du processeur vidéo à charger pour exécuter la commande :

R32	SX7	SX6	SX5	SX4	SX3	SX2	SX1	SX0	abscisse source
R33	0	0	0	0	0	0	0	SX8	(0 à 511)

SX0 à SX8 : abscisse du pixel origine

R34	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	ordonnée source
R35	0	0	0	0	0	0	SY9	SY8	(0 à 1023)

SY0 à SY9 : ordonnée du pixel origine.

R45	0	0	0	MXS	0	0	0	0	paramètres

MXS : Mémoire vidéo de destination

0 pour sélectionner la VRAM principale.

1 pour accéder à la mémoire vidéo étendue sur les MSX2 avec 192 Ko de VRAM.

R46	0	1	0	0	LO3	LO2	LO1	LO0	commandes

LO0 à LO3 : opérateur logique à appliquer

0000 = direct 1000 = Tdirect

0001 = AND 1001 = TAND

0010 = OR 1010 = TOR

0011 = XOR 1011 = TXOR

0100 = NOT 1100 = TNOT

Exécution : pour exécuter l'instruction POINT, voici la marche à suivre :

1 - charger les registres ci-dessus sauf le 46

2 - mettre à 0100B (04H) les 4 bits de poids fort du registre 46, charger les 4 bits de poids faible avec le code de l'opérateur logique désiré.

3 - lire le registre d'état 2 (status), tester l'état du bit CE. Si celui-ci est à 1, alors l'instruction n'est pas terminée, recommencer l'étape 3. Lorsque le bit CE passe à 1, on peut poursuivre avec l'étape 4.

4 - lire le registre d'état 7, il contient le code de la couleur du pixel.

Etat : Une fois l'instruction terminée, les registres su processeur vidéo se trouvent dans l'état suivant :

32-33	34-35	36-37	38-39	40-41	42-43	44	45
idem	idem					code	idem

5.9 LES DIFFERENTS MODES D'AFFICHAGE (SCREEN 0 A SCREEN 8)

Nous allons voir exactement comment fonctionne chaque mode d'affichage :

<u>SCREEN 0</u> 40 colonnes MSX1/MSX2

Résolution :

Texte : 6 x 8 / 40 caractères sur 24 lignes

Couleurs : 2 couleurs parmi 15 (MSX1) ou parmi 512 (MSX2)

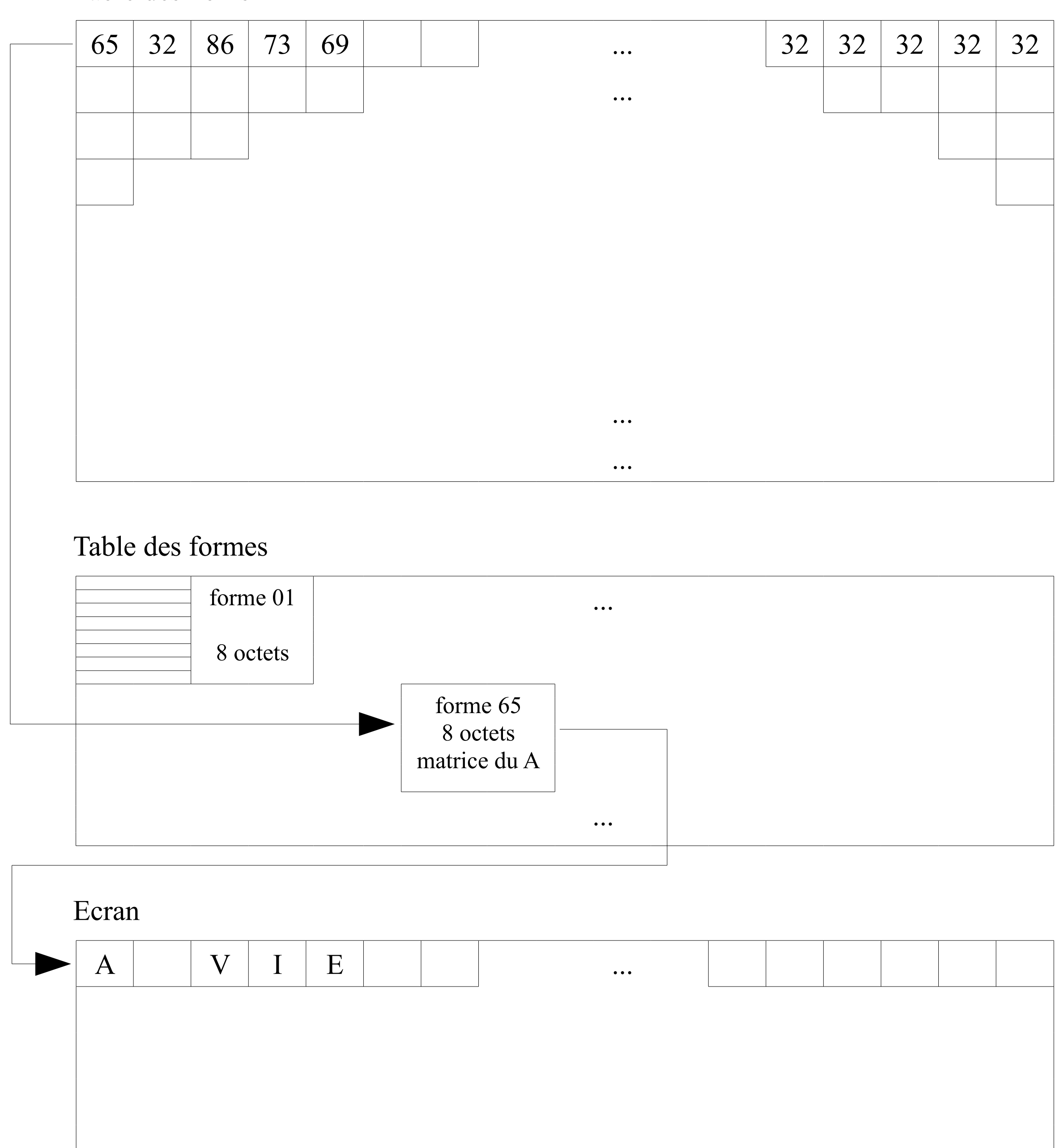
Sprites :

Taille d'une page écran : 4 Ko

Nombre de pages maximum : 32 avec 128 Ko de VRAM

Table des noms :	00000H - 003BFH	960 octets
Table des couleurs :		
Table des formes :	00800H - 00FFFH	2048 octets
Table des attributs sprite :		
Table des formes de sprite :		
Table des couleurs sprite :		
Table de palette (MSX2) :	00400H - 0041FH	32 octets

Voici le schéma de fonctionnement du mode texte (les nombres sont tous en décimal) :



La table des noms contient un octet par caractère à l'écran, soit 40 colonnes fois 24 lignes, soit 960 caractères donc 960 octets. Chaque octet renferme le code ASCII du caractère à afficher à l'écran. Ainsi, si à l'écran se trouvent uniquement les caractères « A VIE » dans le coin supérieur gauche, la table des noms ressemblera à :

00000H	65	ou 041H, code du « A »
00001H	32	ou 020H, code de l'espace
00002H	86	ou 056H, code du « V »
00003H	73	ou 049H, code du « I »
00004H	69	ou 045H, code du « E »

Le reste de la table serait alors rempli de 32 (020H), code de l'espace (ou éventuellement de 0).

La table des formes contient, elle, 2048 octets regroupés par paquets de 8 octets. Ce qui donne 2048 divisé par 8, soit 256 paquets. Chaque paquet correspond à un caractère (code ASCII) et en définit la forme. Par exemple le 65ème paquet code le caractère « A » et ressemble à ceci :

Paquet n°65, donc octet 65*8 = 520 (208H). On trouvera donc l'information qui nous intéresse à l'adresse de début de la table des formes (00800H) plus 208H, soit 00A08H.

00A08H	contient	32	ou	020H	or	020H	=	00100000B			X		
00A09H	contient	80	ou	050H	or	050H	=	01010000B		X		X	
00A0AH	contient	136	ou	088H	or	088H	=	10001000B	X				X
00A0BH	contient	136	ou	088H	or	088H	=	10001000B	X				X
00A0CH	contient	248	ou	0F8H	or	0F8H	=	11111000B	X	X	X	X	X
00A0DH	contient	136	ou	088H	or	088H	=	10001000B	X				X
00A0EH	contient	136	ou	088H	or	088H	=	10001000B	X				X
00A0FH	contient	0	ou	000H	or	000H	=	00000000B					

Les deux bits de poids faible doivent toujours rester à 0 et ne sont de toute manière jamais pris en compte (car tout caractère se trouve défini dans une matrice 6 sur 8).

Redéfinissons à présent le caractère « A », faisons la démarche en sens inverse

X	X	X	X	X	X		11111100B	=	0FCH	
X		X			X		10100100B	=	0A4H	
X					X	donc en	10000100B	=	084H	soit les valeurs
X		X			X		10100100B	=	0A4H	
X		X			X	binaire	10100100B	=	0A4H	en hexadécimal
X			X		X		10010100B	=	094H	
X	X			X	X		11001100B	=	0CCH	
X		X	X		X		10110100B	=	0B4H	

Il suffit à présent d'exécuter le petit programme suivant :

```
10 SCREEN 0 : WIDTH 40 : CLS : AD=&HA08
20 FOR I = 0 TO 7 : READ A$ : VPOKE AD+I, VAL(« &H »+ A$) : NEXT I
30 PRINT « A A A A »
50 DATA FC, A4, 84, A4, A4, 94, CC, B4
```

Tous les « A » de l'écran se transforment en signes cabalistiques d'un ésotérisme certain. Un simple SCREEN 0 fera revenir les choses à la normale.

<u>SCREEN 0</u>   80 colonnes   MSX2

Résolution :	
Texte :	6 x 8 / 80 caractères sur 24 ou 26,5 lignes
Couleurs :	4 couleurs parmi 512
Sprites :	

Taille d'une page écran :	8 Ko
Nombre de pages maximum :	16 avec 128 Ko de VRAM

Table des noms 24 lignes :	00000H - 0077FH	1920 octets
26,5 lignes :	00000H - 0086FH	2160 octets
Table des couleurs 24 lignes :	00800H - 008EFH	240 octets
26,5 lignes :	00800H - 0090DH	270 octets
Table des formes :	01000H - 017FFH	2048 octets
Table des attributs sprite :		
Table des formes de sprite :		
Table des couleurs sprite :		
Table de palette (MSX2) :	00F00H - 00F1FH	32 octets

Le mode 80 colonnes fonctionne de manière identique au mode texte 40 colonnes (voir ci-dessus pour plus de précisions). La table des noms content toujours un octet par caractère à l'écran, soit cette fois 80 colonnes fois 24 lignes donc 1920 octets, ou 80 colonnes fois 26,5 lignes (allez, on arrondit à 27), soit 2160 octets. Chaque octet renferme le code ASCII du caractère à afficher à l'écran. Ainsi, si à l'écran se trouvent uniquement les caractères « A EVE » dans le coin supérieur gauche, la table des noms ressemblera à :

00000H	65	ou 41H, code du « A »
00001H	32	ou 020H, code de l'espace
00002H	69	ou 045H, code du « E »
00003H	86	ou 056H, code du « V »
00004H	69	ou 045H, code du « E »

Le reste de la table serait alors rempli de 32 (020H), code de l'espace (ou éventuellement de 0).

La table des formes contient, elle, 2048 octets regroupés par paquets de 8 octets. Ce qui donne 2048 divisé par 8, soit 256 paquets. Chaque paquet correspond à un caractère (code ASCII) et en définit la forme. Par exemple le 66ème paquet code le caractère « B » et ressemble à ceci :

Paquet n°66, donc octet 66*8 = 528 (210H). On trouvera donc l'information qui nous intéresse à l'adresse de début de la table des formes (01000H) plus 210H, soit 01210H.

01210H	contient	240	ou	0F0H	or	0F0H	=	11110000B	X X X X
01211H	contient	72	ou	048H	or	048H	=	01001000B	X X
01212H	contient	72	ou	048H	or	048H	=	01001000B	X X
01213H	contient	112	ou	070H	or	070H	=	01110000B	X X X
01214H	contient	72	ou	048H	or	048H	=	01001000B	X X
01215H	contient	72	ou	048H	or	048H	=	01001000B	X X
01216H	contient	240	ou	0F0H	or	0F0H	=	11110000B	X X X X
01217H	contient	0	ou	000H	or	000H	=	00000000B	

Les deux bits de poids faible doivent toujours rester à 0 et ne sont de toute manière jamais pris en compte (car tout caractère se trouve défini dans une matrice 6 sur 8).

Redéfinissons à présent le caractère « B », faisons la démarche en sens inverse

X X X X X		11111000B	=	0F8H	
X X		10001000B	=	088H	
X X X X	donc en	10111000B	=	0B8H	soit les valeurs
X X X		10011000B	=	098H	
X X X X	binaire	10111000B	=	0B8H	en hexadécimal
X X		10001000B	=	088H	
X X		10001000B	=	088H	
X X X X X		11111000B	=	0F8H	

Il suffit à présent d'exécuter le petit programme suivant :

```
10 SCREEN 0 : WIDTH 80 : CLS : AD=&H1210
20 FOR I = 0 TO 7 : READ A$ : VPOKE AD+I, VAL(« &H »+ A$) : NEXT I
30 PRINT « B B B B »
50 DATA F8, 88, B8, 98, B8, 88, 88, F8, 00
```

Tous les « B » de l'écran se transforment en petit signe « E » en vidéo inverse. L'instruction SCREEN 0 fera revenir les choses à la normale.

En ce qui concerne les possibilités de faire clignoter le texte dans ce mode, voir le paragraphe sur les registres du processeur vidéo (registres 12 et 13).

SCREEN 1 32 colonnes MSX1/MSX2

Résolution :	
Texte :	8 x 8 / 32 caractères sur 24 lignes
Couleurs :	16 couleurs parmi 512
Sprites :	type 1

Taille d'une page écran :	4 Ko
Nombre de pages maximum :	32 avec 128 Ko de VRAM

Table des noms :	01800H - 01AFFH	768 octets
Table des couleurs :	02000H - 0201FH	32 octets
Table des formes :	00000H - 007FFH	2048 octets
Table des attributs sprite :	01B00H - 01B7FH	128 octets
Table des formes de sprite :	03800H - 03FFFH	2048 octets
Table des couleurs sprite :		
Table de palette (MSX2) :	02020H - 0203FH	32 octets

Le mode 32 colonnes fonctionne de manière identique au mode texte 40 colonnes (voir ci-dessus pour plus de précisions). La table des noms content toujours un octet par caractère à l'écran, soit cette fois 32 colonnes fois 24 lignes donc 768 octets. Chaque octet renferme le code ASCII du caractère à afficher à l'écran. Ainsi, si à l'écran se trouvent uniquement les caractères « A BEN » dans le coin supérieur gauche, la table des noms ressemblera à :

01800H	65	ou 41H, code du « A »
01801H	32	ou 020H, code de l'espace
01802H	66	ou 042H, code du « B »
01803H	69	ou 045H, code du « E »
01804H	78	ou 04EH, code du « N »

Le reste de la table serait alors rempli de 32 (020H), code de l'espace (ou éventuellement de 0).

La table des formes contient, elle, 2048 octets regroupés par paquets de 8 octets. Ce qui donne 2048 divisé par 8, soit 256 paquets. Chaque paquet correspond à un caractère (code ASCII) et en définit la forme. Par exemple le 32ème paquet code le caractère espace et ressemble à ceci :

Paquet n°32, donc octet 32*8 = 256 (100H). On trouvera donc l'information qui nous intéresse à l'adresse de début de la table des formes (00000H) plus 100H, soit 0100H. Les cases mémoire vidéo de 00100H à 00107H sont toutes remplies avec des 0 (espace = rien). Un simple VPOKE &H100, 255 devrait vous convaincre de l'utilité de bien connaître le processeur vidéo.

Notez que dans ce mode, tous les bits peuvent être utilisés pour définir un caractère (voir les modes

40 et 80 colonnes) car les caractères se trouvent cette fois définis dans une matrice 8 sur 8).

Essayez donc le programme Basic suivant (tout à fait désopilant) :

```
10 SCREEN 1 : WIDTH 32
20 LOCATE 12, 12 : PRINT « COUCOU »
30 FOR J = 0 TO 7
40 FOR I = 0 TO 7
50 VPOKE &H100+I, 2^J
60 NEXT I, J
70 GOTO 30
```

Ce que l'on f ait avec 7 lignes Basic, c'est fou ! Après un BREAK, l'instruction SCREEN 1 fera revenir les choses à la (triste) normale.

Pour ce qui est des couleurs, rien de bien affriolant dans ce mode. Les 32 octets de la table des couleurs codent chacun la couleur de huit caractères suivant leur code ASCII. Ce qui signifie par exemple que le contenu de 02009H donne la couleur des lettres H à O, sachant que les 4 bits de poids fort codent la couleur du texte alors que les quatre bits de poids faible codent la couleur du fond. Le seul moyen d'utiliser un tel système est de redéfinir entièrement le jeu de caractères? A part cela, on peut toujours s'amuser à changer la couleur du curseur (code 255 ou 0FFH) :

```
10 SCREEN 1 : WIDTH 32 : COLOR 10, 0
20 VPOKE &H201F, &HD7
```

donne un curseur mauve (0DH) avec des caractères cyan (07H).

Après toutes ces émotions, et suivant l'adage « All work and no play makes Jack a sad boy », je vous propose un petit jeu. Profitez-en, amusez-vous bien, et hop (fonctionne sur MSX2 et MSX1 en retirant les instructions COLOR =) :

```
10 GOTO 90
20 I = STICK(0) : IF I = 3 THEN X = X + DD : GOTO 50 ELSE IF I = 7 THEN X = X - DD :
GOTO 50 ELSE 50
30 FOR I = 0 TO 7 : VPOKE &H208 + I, 128 : NEXT : FOR I = 0 TO 7 : VPOKE &H210 + I, 1 :
NEXT : T = 3 : X = 40 : Y = 88 : PUT SPRITE 1, (X, Y), 9 : COLOR 10, 2 : FOR I = 0 TO 23 :
PRINTTAB(T-1) « B »+ STRING$ (N+1, « O ») + « A »: NEXT : VPOKE &H2004, &HC2 :
VPOKE &H2009, 17 : VPOKE &H2006, &HF2 : COLOR = (2, 1, 1, 2)
40 LOCATE 20, 5 : PRINT « Attention » : FOR J = 3 TO 1 STEP -1 : LOCATE 23, 7 : PRINT J :
PLAY « L8D » : FOR I = 1 TO 800 : NEXT I, J : LOCATE 20, 5 : PRINT SPACE$(9) : LOCATE
24, 7 : PRINT «  » : LOCATE 0, 24 : PLAY « L2O6A »
50 SC = SC +1 : PUT SPRITE 1, (X, Y), 8 : I = &H1960+X/8 : IF VPEEK (I)<>79 OR VPEEK
(I+1)<>79 THEN 100 ELSE D = INT(RND(14)*3) : D = -D*(T>2 AND T + N<29)-2*(T<=2)-
(T+N>=29) : ON D GOTO 70, 80
60 PRINT TAB (T-1) « B »+STRING$(N+1, « O »)+ « A » : GOTO 20
70 T=T-1 : PRINT TAB (T) « / »+STRING$(N, « O »)+ « / » : GOTO 20
80 T=T+1 : PRINT TAB (T-1) « \ »+STRING$(N, « O »)+ « \ » : GOTO 20
90 SCREEN 1 : WIDTH 32 : COLOR 10, 1 : SPRITE$(1) = « I »+CHR$(127)+ « I »+CHR$(8)+
« I »+CHR$(127)+ « I »+CHR$(8) : PRINT : PRINT : FOR I = 1 TO 4 : PRINT « - RALLY - »;:
NEXT : LOCATE 5,12 :INPUT « NIVEAU (1 ou 2) »; NN : NN=NN+1 : IF NN=3 THEN N=4 :
```

DD=4 : GOTO 30 : ELSE N=8 : DD=2 : GOTO 30
100 VPOKE &H2007, &HF2 : VPOKE &H200A, &H72 : VPOKE &H200C, &H72 : VPOKE &H200D, &H72 : VPOKE &H200E, &H72 : LOCATE 1, 23 : PRINT « Score = »SC*10; : FOR I = 1 TO 100000! : NEXT

Voici l'exemple parfait de ce que l'on peut faire pour sa petite voisine par un après-midi pluvieux, gris.

SCREEN 2 MSX1/MSX2

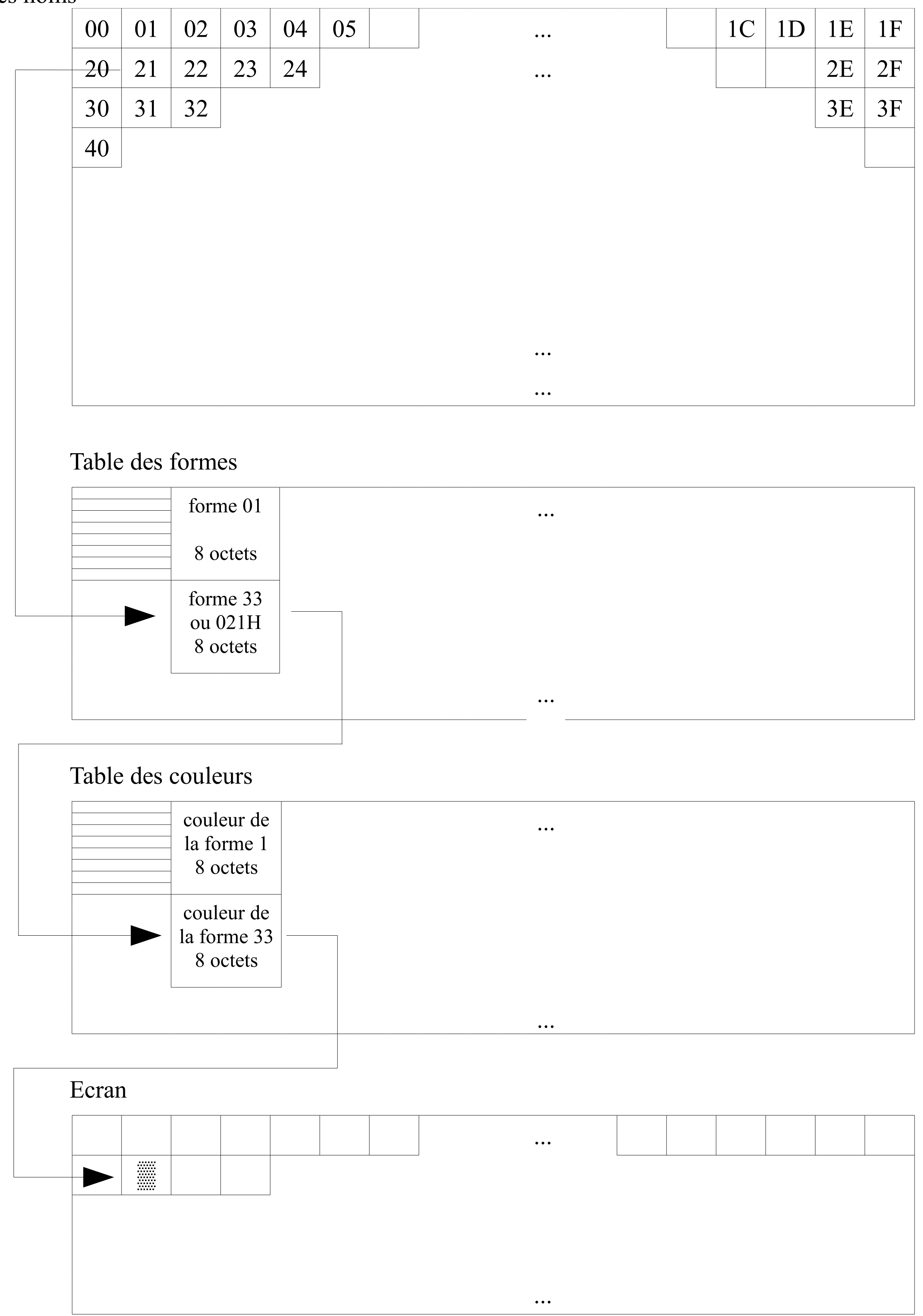
Résolution :	256 sur 192 contraintes de couleur
Texte :	8 x 8 / 32 caractères sur 24 lignes
Couleurs :	15 couleurs (MSX1)
	16 couleurs parmi parmi 512 (MSX2)
Sprites :	type 1

Taille d'une page écran :	16 Ko
Nombre de pages maximum :	8 avec 128 Ko de VRAM

Table des noms :	01800H - 01AFFH	768 octets
Table des couleurs :	02000H - 037FFH	6144 octets
Table des formes :	00000H - 017FFH	6144 octets
Table des attributs sprite :	01B00H - 01B7FH	128 octets
Table des formes de sprite :	03800H - 03FFFH	2048 octets
Table des couleurs sprite :		
Table de palette (MSX2) :	02020H - 0203FH	32 octets

Le mode d'écran 2 ressemble beaucoup au mode texte dans son utilisation.

Voici le schéma de fonctionnement du mode graphique SCREEN 2 (les nombres sont tous en hexadécimal) :



La table des noms est constituée de 768 octets. Le processeur remplit cette table avec des nombres de 0 à 255 3 fois (3*256 = 768). L'écran se trouve donc divisé en trois tiers. Ainsi, le programmeur peut travailler sur une seule table, la table des formes. Cette dernière contient 6144 octets, soit 2048 octets par tiers d'écran. 8 octets codant un caractère, on trouve bien 2048/8 = 256 caractères par tiers

d'écran. La carte mémoire de la table des formes par rapport à l'écran serait :

16 pixels

LIGNES 0 à 7

00000H	00008H	00010H	00018H	00020H
00001H	00009H	00011H		
00002H	0000AH			
00003H	0000BH			
00004H	0000CH			
00005H	0000DH			
00006H	0000EH			
00007H	0000FH			
00100H	00108H			
00101H	00109H			

La table des couleurs est exactement la même, mais à partir de l'adresse 002000H.

Dans les deux tables (formes et couleurs), un octet code huit pixels. Dans la table des formes, chaque bit indique si le pixel équivalent est de couleur 0 ou de couleur 1. Dans la table des couleurs, les 4 bits de poids fort donnent la couleur des pixels à 1 (0-15), alors que les 4 bits de poids faible codent la couleur des pixels à 0 (0-15). Par exemple si l'adresse 00009H contient 081H alors que 02009H renferme 0D1H, alors les pixels de coordonnées (8, 1) et (15, 1) seront allumés en magenta (0DH) alors que les pixels (9, 1) à (14, 1) resteront (ou deviendront) noirs.

Bien entendu, rien n'empêche le programmeur averti de modifier la table des noms. Il suffit de faire attention à quel tiers d'écran on désire accéder. Voyons un petit exemple en Basic :

```
10 SCREEN 2 : COLOR 10, 1, 1 : CLS
20 '
30 ' TABLE DES FORMES, FORME N°10
40 FOR I = 0 TO 7 : READ A$ : VPOKE I, VAL (« &H » + A$) : NEXT
50 '
60 ' TABLE DES COULEURS, FORME N°0
70 FOR I = &H2000 TO &H2007 : READ A$ : VPOKE I, VAL (« &H »+ A$) : NEXT
80 '
90 ' TABLE DES NOMS, JOLI CADRE EN BRIQUES
100 FOR I = &H1800 TO &H181F : VPOKE I, 0 : NEXT
110 FOR I = &H18E0 TO &H18FF : VPOKE I, 0 : NEXT
120 FOR I = &H1800 TO &H18E0  STEP 32 : VPOKE I, 0 : NEXT
130 FOR I = &H181F TO &H18FF  STEP 32 : VPOKE I, 0 : NEXT
140 '
150 CIRCLE (127, 98), 50, 14 : PAINT (127, 98), 14
160'
170 GOTO 170
180 '
190 ' DONNÉES POUR LA FORME
200 DATA 00, BB, 00, DD, 00, BB, 00, DD
```

```
210 '
220 ' DONNÉES POUR LES COULEURS
230 DATA 00, 60, 00, 50, 00, A0, 00, 20
```

Nous avons redéfini la forme 0, puis nous l'avons affichée sur tout la première et la septième ligne, ainsi que sur les côtés. Essayez donc de faire la même chose avec le Basic classique.

Vous remarquerez, à la ligne 150, que tous les dessins s'effectuent derrière notre cadre (ce qui est logique mais agréable). Attention cependant à ne pas utiliser les pixels dans le cadre entre (0, 0) et (7, 7).

<u>SCREEN 3</u> MSX1/MSX2

Résolution :	64 sur 48, pas de contrainte
Texte :	32 x 32 par 4 pixels/ 8 caractères sur 6 lignes
Couleurs :	15 couleurs (MSX1)
	16 couleurs parmi parmi 512 (MSX2)
Sprites :	type 1

Taille d'une page écran :	4 Ko
Nombre de pages maximum :	32 avec 128 Ko de VRAM

Table des noms :	00800H - 00AFFH	768 octets
Table des couleurs :		
Table des formes :	00000H - 007FFH	2048 octets
Table des attributs sprite :	01B00H - 01B7FH	128 octets
Table des formes de sprite :	03800H - 03FFFH	2048 octets
Table des couleurs sprite :		
Table de palette (MSX2) :	02020H - 0203FH	32 octets

Le mode d'écran 3 fonctionne comme le mode 2 (voir le SCREEN 2 pour plus de précisions), mais avec un écran de 64 sur 48. En matière de taille, un pixel mode 3 équivaut à 4 pixels mode 2.

0, 0 0, 3	4, 0 7, 3	8, 0	15, 3	4 « gros » pixels

0, 4 0, 7	4, 4 7, 7	8, 4	15, 7	4 « gros » pixels

Dans la table des formes, chaque octet code 2 gros pixels. Les 4 bits de poids fort donnent la couleur (0 à 15) du pixel d'abscisse paire, alors que les 4 bits de poids faible indiquent la couleur (0 à 15) du pixel d'abscisse impaire.

Si l'on néglige la table des noms, on ne travaille que sur la table des formes qui ressemble à :

00000H	0, 0	63, 0
00100H	0, 7	63, 7
	0, 8	63, 8
00200H	0, 15	63, 15
	0, 16	63, 16
00300H	0, 23	63, 23
	0, 24	63, 24
00400H	0, 31	63, 31
	0, 32	63, 32
00500H	0, 39	63, 39
	0, 40	63, 40
	0, 47	63, 47

Il est alors facile d'adresser n'importe quel pixel, comme le montre l'exemple suivant en Basic :

```
10 SCREEN 3 : COLOR 10, 1, 1 : CLS
20 '
30 X = 25 : Y = 117 : C = 6 : PSET (X, Y), 7 : FOR I = 1 TO 1000 : NEXT
40 VPOKE (INT(Y/32) * 256) + INT(X/8)*8 + (Y MOD 32)/4, -(C*16)*(X MOD 4 < 2) - C*(X MOD 4 > 1)
140 GOTO 140
```

Pour ceux qui désirent profiter des possibilités offertes par la table des noms, sachez que chaque octet de cette table contient le numéro de la forme à utiliser. Seulement, on n'utilise que deux lignes de la forme suivant la règle :

lignes 0 et 1 --- VRAM de 00800H à 0081FH

lignes 2 et 3 --- VRAM de 00820H à 0083FH

lignes 4 et 5 --- VRAM de 00840H à 0085FH

lignes 6 et 7 --- VRAM de 00860H à 0087FH

lignes 0 et 1 --- VRAM de 00880H à 0089FH

lignes 2 et 3 --- VRAM de 008A0H à 008BFH

et ainsi de suite ...

Voici un nouvel exemple Basic utilisant cette technique :

```
10 SCREEN 3 : COLOR 10, 1, 1 : CLS
20 '
30 ' TABLE DES FORMES, FORME N°0
40 VPOKE 0, &HA8 : VPOKE 1, &H8A : VPOKE 2, &HA8 : VPOKE 3, &H8A : VPOKE 4,
&HA8 : VPOKE 5, &H8A : VPOKE 6, &HA8 : VPOKE 7, &H8A
50 '
60 ' TABLE DES NOMS, JOLI CADRE EN BRIQUES
70 FOR I = 0 TO 31 : VPOKE &H800 + I, 0 : NEXT
80 FOR I = 0 TO 31 : VPOKE &HA00 + I, 0 : NEXT
90 FOR I = &H800 TO &HA00 STEP 32 : VPOKE I, 0 : NEXT
100 FOR I = &H81F TO &HA1F STEP 32 : VPOKE I, 0 : NEXT
110 '
120 CIRCLE (127, 98) , 50, 14 : PAINT (127, 98), 14
130 '
140 GOTO 140
```

Notez qu'en SCREEN 3, lorsque vous utilisez les instructions Basic OPEN « GRP: » AS #1 puis PRINT #1, « Texte » le texte s'affiche en grandes lettres (4 fois la taille normale 40 colonnes).

SCREEN 4 MSX2

Résolution :	256 sur 192 contraintes de couleur
Texte :	8 x 8 / 32 caractères sur 24 lignes
Couleurs :	16 couleurs parmi parmi 512
Sprites :	type 2

Taille d'une page écran :	16 Ko
Nombre de pages maximum :	8 avec 128 Ko de VRAM

Table des noms :	01800H - 01AFFH	768 octets
Table des couleurs :	02000H - 037FFH	6144 octets
Table des formes :	00000H - 017FFH	6144 octets
Table des attributs sprite :	01E00H - 01E7FH	128 octets
Table des formes de sprite :	03800H - 03FFFH	2048 octets
Table des couleurs sprite :	01C00H - 0D7FFH	512 octets
Table de palette :	01E80H - 01E9FH	32 octets

Le SCREEN 4 est en tous points identique au SCREEN 2 à l'exception des sprites qui sont de type 2. Voir le paragraphe concernant le SCREEN 2 pour plus d'information sur l'affichage. Pour de plus amples renseignements sur les sprites, voir le paragraphe 5.10 « Les sprites et leur fonctionnement ».

SCREEN 5 MSX2

Résolution :	256 sur 192 bitmap
	256 sur 212 bitmap
Texte :	8 x 8 / 32 caractères sur 24 ou 26,5 lignes
Couleurs :	16 couleurs parmi parmi 512
Sprites :	type 2

Taille d'une page écran :	32 Ko
Nombre de pages maximum :	4 avec 128 Ko de VRAM

Table des noms 192 lignes :	00000H - 05FFFH	24576 octets
212 lignes :	00000H - 069FFH	27136 octets
Table des couleurs :		
Table des formes :		
Table des attributs sprite :	07600H - 0767FH	128 octets
Table des formes de sprite :	07800H - 07FFFH	2048 octets
Table des couleurs sprite :	07400H - 075FFH	512 octets
Table de palette :	07680H - 0769FH	32 octets

Avec le mode d'écran 5, on entre dans le domaine des graphismes en « bitmap ». Ce terme signifie que chaque pixel est codé directement par sa couleur, et ce, indépendamment des autres pixels. Il n'existe donc plus de contrainte de couleur, ni même de table de couleur ou de formes. Ce mode « bitmap » apparaît donc comme idéal puisqu'il allie puissance et simplicité.

Dans ce mode, chaque octet de la table des noms code deux pixels. Les 4 bits de poids fort donnent la couleur (0-15) du pixel d'abscisse paire, alors que les 4 bits de poids faible donnent la couleur (0-15) du pixel d'abscisse impaire.

Le registre 2 du V9938 indique la page à afficher (0 à 3) :

R2	0	A16	A15	1	1	1	1	1

A16 et A15 : page à afficher 0 à 3.

SCREEN 6 MSX2

Résolution :	512 sur 192 bitmap
	512 sur 212 bitmap
Texte :	8 x 8 / 64 caractères sur 24 ou 26,5 lignes
Couleurs :	4 couleurs parmi parmi 512
Sprites :	type 2

Taille d'une page écran :	32 Ko
Nombre de pages maximum :	4 avec 128 Ko de VRAM

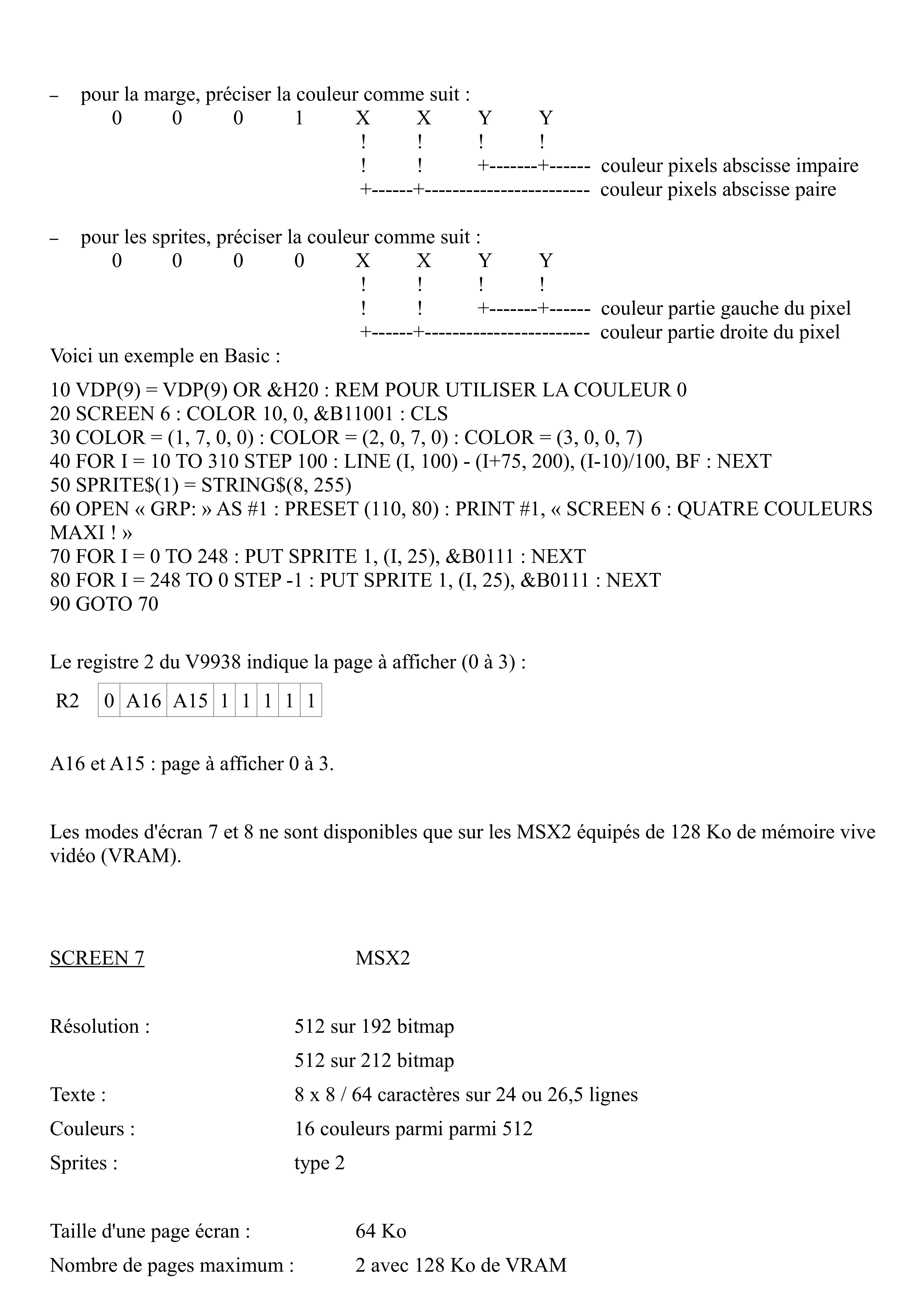
Table des noms 192 lignes :	00000H - 05FFFH	24576 octets
212 lignes :	00000H - 069FFH	27136 octets
Table des couleurs :		
Table des formes :		
Table des attributs sprite :	07600H - 0767FH	128 octets
Table des formes de sprite :	07800H - 07FFFH	2048 octets
Table des couleurs sprite :	07400H - 075FFH	512 octets
Table de palette :	07680H - 0769FH	32 octets

En mode d'écran 6, chaque octet de la table des noms code 4 pixels. Les bits 6 et 7 (poids fort) donnent la couleur (0-3) du pixel multiple de 4. Les bits 4 et 5 codent la couleur du pixel immédiatement à la droite du pixel précédent, et ainsi de suite...

La table des noms ressemble donc à :

poids fort	7	6	5	4	3	2	1	0	poids faible
00000H	(0,	0)	(1,	0)	(2,	0)	(3,	0)	
00001H	(4,	0)	(5,	0)	(6,	0)	(7,	0)	
	...							...	
	...							...	
0007FH	(508,	0)	(509,	0)	(510,	0)	(511,	0)	
00080H	(0,	1)	(1,	1)	(2,	1)	(3,	1)	

Le processeur vidéo ne peut afficher que 4 couleurs en SCREEN 6. C'est vrai, mais tout le monde sait qu'en juxtaposant deux couleurs, avec des pixels suffisamment petits, on obtient une troisième couleur. Les concepteurs du V9938 ont conçu une instruction qui mélange automatiquement deux couleurs (« hardware tiling »). Cette fonction peut s'appliquer à la marge (ainsi, 7 couleurs de marge sont possibles) et surtout aux sprites de la manière suivante :



– pour la marge, préciser la couleur comme suit :

```
0       0       0       1       X       X       Y       Y
                                !       !       !       !
                                !       !       +-------+------  couleur pixels abscisse impaire
                                +------+-----------------------  couleur pixels abscisse paire
```

– pour les sprites, préciser la couleur comme suit :

```
0       0       0       0       X       X       Y       Y
                                !       !       !       !
                                !       !       +-------+------  couleur partie gauche du pixel
                                +------+-----------------------  couleur partie droite du pixel
```

Voici un exemple en Basic :

```
10 VDP(9) = VDP(9) OR &H20 : REM POUR UTILISER LA COULEUR 0
20 SCREEN 6 : COLOR 10, 0, &B11001 : CLS
30 COLOR = (1, 7, 0, 0) : COLOR = (2, 0, 7, 0) : COLOR = (3, 0, 0, 7)
40 FOR I = 10 TO 310 STEP 100 : LINE (I, 100) - (I+75, 200), (I-10)/100, BF : NEXT
50 SPRITE$(1) = STRING$(8, 255)
60 OPEN « GRP: » AS #1 : PRESET (110, 80) : PRINT #1, « SCREEN 6 : QUATRE COULEURS
MAXI ! »
70 FOR I = 0 TO 248 : PUT SPRITE 1, (I, 25), &B0111 : NEXT
80 FOR I = 248 TO 0 STEP -1 : PUT SPRITE 1, (I, 25), &B0111 : NEXT
90 GOTO 70
```

Le registre 2 du V9938 indique la page à afficher (0 à 3) :

R2	0	A16	A15	1	1	1	1	1

A16 et A15 : page à afficher 0 à 3.

Les modes d'écran 7 et 8 ne sont disponibles que sur les MSX2 équipés de 128 Ko de mémoire vive vidéo (VRAM).

<u>SCREEN 7</u> MSX2

Résolution : 512 sur 192 bitmap

512 sur 212 bitmap

Texte : 8 x 8 / 64 caractères sur 24 ou 26,5 lignes

Couleurs : 16 couleurs parmi parmi 512

Sprites : type 2

Taille d'une page écran : 64 Ko

Nombre de pages maximum : 2 avec 128 Ko de VRAM

Table des noms 192 lignes :	00000H - 0BFFFH	49152 octets
212 lignes :	00000H - 0D3FFH	54272 octets
Table des couleurs :		
Table des formes :		
Table des attributs sprite :	0FA00H - 0FA7FH	128 octets
Table des formes de sprite :	0F000H - 0F7FFH	2048 octets
Table des couleurs sprite :	0F800H - 0F9FFH	512 octets
Table de palette :	0FA80H - 0FA9FH	32 octets

Dans ce mode, chaque octet de la table des noms code deux pixels. Les 4 bits de poids fort donnent la couleur (0-15) du pixel d'abscisse paire, alors que les 4 bits de poids faible donnent la couleur (0-15) du pixel d'abscisse impaire.

Le registre 2 du V9938 indique la page à afficher (0 à 3) :

R2	0	0	A16	1	1	1	1	1

A16 et A15 : page à afficher 0 à 1.

SCREEN 8 MSX2

Résolution :	256 sur 192 bitmap
	256 sur 212 bitmap
Texte :	8 x 8 / 32 caractères sur 24 ou 26,5 lignes
Couleurs :	256 couleurs
Sprites :	type 2

Taille d'une page écran :	64 Ko
Nombre de pages maximum :	2 avec 128 Ko de VRAM

Table des noms 192 lignes :	00000H - 0BFFFH	49152 octets
212 lignes :	00000H - 0D3FFH	54272 octets
Table des couleurs :		
Table des formes :		
Table des attributs sprite :	0FA00H - 0FA7FH	128 octets
Table des formes de sprite :	0F000H - 0F7FFH	2048 octets
Table des couleurs sprite :	0F800H - 0F9FFH	512 octets

Table de palette : 0FA80H - 0FA9FH 32 octets

Dans ce mode, chaque octet de la table des noms code un seul pixel. Les huit bits donnent la couleur (0-255) du pixel.

La couleur est déterminée de la manière suivante :

octet	G2	G1	G0	R2	R1	R0	B1	B0

G0 à G2 : intensité de vert (0-7)

R0 à R2 : intensité de rouge (0-7)

B0 et B1 : intensité de bleu (0-3)

En Basic, on peut calculer le numéro de la couleur désirée avec la formule :

100 C = 32*G + 4*R + B avec G et R de 0 à 7, B entre 0 et 3.

Le registre 2 du V9938 indique la page à afficher (0 à 3) :

R2	0	0	A16	1	1	1	1	1

A16 et A15 : page à afficher 0 à 1.

5.10 LES SPRITES ET LEUR FONCTIONNEMENT

Les sprites, ou lutins graphiques, sont des motifs graphiques pouvant être affichés automatiquement à n'importe quel pixel dans n'importe quelle couleur, ceci en se superposant aux dessins dans l'écran, sans les effacer. Ces facilités les prédestinent au mouvement puisqu'il suffut de modifier deux octets (les coordonnées du sprite) pour faire déplacer tout un motif sans toucher au reste de l'écran. Le V9938 peut mémoriser jusqu'à 256 sprites, il peut en afficher simultanément au maximum 32. Les sprites se trouvent définis dans un matrice 8 sur 8 ou 16 sur 16. Ils peuvent être de 2 tailles, normale ou double.

Attention : les coordonnées des coins supérieurs ne sont pas (0, 0) et (255, 0), mais bien (0, 255) et (255, 255). On gagne donc un pixel par rapport à l'écran normal. Pour vous en persuader, essayez donc ce petit programme...

```
10 SCREEN 7 : COLOR 10, 0, 0
20 LINE (0, 0) - (511, 0), 15
30 SPRITE$(0) = CHR$ (&HFF)
40 FOR I = 0 TO 255
50 PUT SPRITE 0, (I, 0), 8
60 NEXT
70 GOTO 40
```

... puis celui-ci :

```
10 SCREEN 7 : COLOR 10, 0, 0
20 LINE (0, 0) - (511, 0), 15
30 SPRITE$(0) = CHR$ (&HFF)
40 FOR I = 0 TO 255
50 PUT SPRITE 0, (I, 255), 8 : REM ici Y = 255
60 NEXT
70 GOTO 40
```

Il existe deux modes de fonctionnement des sprites bien distincts. Le V9938 choisit automatiquement le mode adéquat suivant le mode écran. En SCREEN 1, 2 et 3, c'est le mode 1 appelé aussi mode restreint (qui correspond au MSX1) alors qu'en SCREEN 4 à 8, c'est le mode 2 ou mode étendu (MSX2).

Le mode 1 (MSX1/MSX2)

Fonctionnement :

Dans ce mode, il peut y avoir 32 sprites à l'écran numérotés de 0 à 31. Plus un sprite a un numéro faible, plus il est prioritaire. Ceci signifie que si le sprite n°3 croise le sprite n°16, on verra le sprite n°3 passer dessus (lorsque des pixels des deux sprites se trouveront aux mêmes coordonnées, c'est les pixels du sprite n°3 qui seront affichés).

4 sprites peuvent, au maximum, se trouver sur la même ligne. A partir du 5ème sprite, toutes les portions communes aux 5 sprites disparaissent sur le (ou les) sprites de plus faible priorité.

Paramètres :

Matrice des sprites - bit 1 (SI) du registre 1 du VDP

SI à 0 = sprites 8 sur 8

SI à 1 = sprites 16 sur 16

Taille des sprites - bit 0 (MAG) du registre 1 du VDP

MAG à 0 = taille normale

MAG à 1 = taille double (16 sur 16, ou 32 sur 32, avec de gros pixels)

Pour bien comprendre le mécanisme de la taille, voici un exemple en Basic :

```
10 SCREEN 2 : COLOR 10, 0, 0 : CLS
20 OPEN « GRP: » AS #1
30 SPRITE$(0) = « ABABABAB »
```

```
40 PRESET (16, 5) : PRINT #1, « UN SPRIT EQUELCONQUE. » : COLOR
8, 1 : PRESET (16, 155) : PRINT #1, « MAG = »
50 LINE (63, 155) - (72, 164), 1 , BF : PRESET (64, 155) : PRINT #1, « 0 »
60 VDP(1) = VDP (1) AND &HFE
70 FOR I = 0 TO 248
80 PUT SPRITE 0, (I, 35), 7
90 NEXT
100 LINE (63, 155) - (72, 164), 1, BF : PRESET (64, 155) : PRINT #1, « 1 »
110 VDP(1) = VDP(1) OR 1
120 FOR I = 0 TO 248
130 PUT SPRITE 0, (I, 35), 7
140 NEXT
150 GOTO 50
```

Collisions :

Lorsque deux sprites entrent en collision (deux pixels ou plus ont les mêmes coordonnées), le bit 5 du registre d'état (status) n°0 passe à 1.

5 sprites :

Lorsque plus de 4 sprites se trouvent sur la même ligne, le bit 6 (5S) du registre d'état (status) n°0 passe à 1. De plus les 5 bits de poids faible du registre d'état n°0 donnent le numéro du 5ème sprite.

Utilisation :

Pour afficher un sprite, il faut d'abord le définir dans la table des formes de sprites.

Pour les sprites 8 sur 8, il suffit de 8 octets pour définir un sprite, sachant qu'un bit représente un pixel, par exemple, en SCREEN 2, la table des formes de sprites pourrait être :

	7 0	huit octets pour le sprite n°0 qui aura cette forme
03800H	11111111	X X X X X X X X
03801H	10100001	X X X
03802H	10010001	X X X
03803H	10001001	X X X
03804H	10000101	X X X
03805H	10000101	X X X
03806H	10001001	X X X
03807H	11111111	X X X X X X X X

Dans le cas de sprites 16 sur 16, les choses deviennent un tout petit peu plus compliquées. Le sprite se décompose en 4 parties :

1	3
2	4

	7 0									
03800H	11111110	X	X	X	X	X	X	X		
03801H	10000010	X						X		
03802H	10001110	X				X	X	X		
03803H	10001000	X				X				
03804H	10001000	X				X				huit octets pour le premier quart du sprite n°0
03805H	10001000	X				X				
03806H	10001000	X				X				
03807H	10001000	X				X				
03808H	10001000	X				X				
03809H	10001000	X				X				
0380AH	10001000	X				X				
0380BH	10001000	X				X				huit octets pour le second quart du sprite n°0
0380CH	10001000	X				X				
0380DH	10001111	X				X	X	X	X	
0380EH	10000000	X								
0380FH	11111111	X	X	X	X	X	X	X	X	
03810H	01111111		X	X	X	X	X	X	X	
03811H	01000001		X						X	
03812H	01110001		X	X	X				X	
03813H	00010001				X				X	huit octets pour le troisième quart du sprite n°0
03814H	00010001				X				X	
03815H	00010001				X				X	
03816H	00010001				X				X	
03817H	00010001				X				X	
03818H	00010001				X				X	
03819H	00010001				X				X	
0381AH	00010001				X				X	
0381BH	00010001				X				X	huit octets pour le dernier quart du sprite n°0
0381CH	00010001				X				X	
0381DH	11110001	X	X	X	X				X	
0381EH	00000001								X	
0381FH	11111111	X	X	X	X	X	X	X	X	

La réunion des quatre parties donne le résultat ci-dessous :

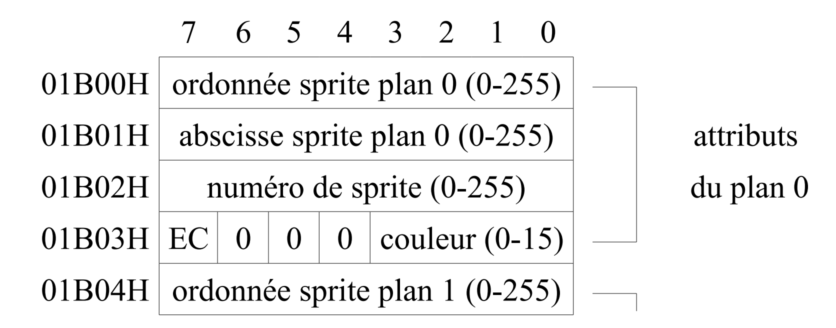
1	3
2	4

```
X X X X X X X     X X X X X X X
X             X   X             X
X       X X X     X X X         X
X       X             X         X
X       X             X         X
X       X             X         X
X       X             X         X
X       X             X         X
X       X             X         X
X       X             X         X
X       X             X         X
X       X             X         X
X       X             X         X
X       X X X X X X X X         X
X                               X
X X X X X X X X X X X X X X X X
```

Une fois le sprite défini, on peut le manipuler à souhait grâce à la table des attributs de sprites.

Cette table fonctionne avec des plans graphiques, et compte 4 octets par plan. On appelle « plan graphique » un espace comprenant un élément graphique. Il y a donc 32 plans (numérotés de 0 à 31) avec chacun un sprite, puis un plan avec les graphismes ordinaires, enfin parfois un plan en entrée vidéo.

La table des attributs de sprites code les caractéristiques de chaque plan sur 4 octets :



	7	6	5	4	3	2	1	0	
01B00H	ordonnée sprite plan 0 (0-255)								attributs du plan 0
01B01H	abscisse sprite plan 0 (0-255)								
01B02H	numéro de sprite (0-255)								
01B03H	EC	0	0	0	couleur (0-15)				
01B04H	ordonnée sprite plan 1 (0-255)								

L'ordonnée peut varier entre 0 et 255. Si le sprite sort de l'écran, le V9938 n'affichera que la partie visible du sprite. En réalité, on utilise les coordonnées de 225 (-31 en complément à 2) à 255, et 0 à 191. Ceci permet de faire entrer un sprite dans l'écran (-31 à -1 ou 225 à 255) ou au contraire de le faire sortir (183 à 191 pour un 8 sur 8) petit à petit.

Notez que si l'ordonnée de n'importe quel sprite est à 208 (0D0H), alors tous les sprites dans les plans supérieurs deviennent invisibles. Par exemple si l'on met l'ordonnée du sprite contenu dans 12 à 208, alors tous les sprites dans les plans 13 à 31 disparaissent.

L'abscisse prend une valeur entre 0 et 255. Pour faire sortir le sprite, pas de problème, on utilise les abscisses entre 247 et 255 (pour un sprite 8 sur 8). Par contre, pour le faire entrer, il faut utiliser le

bit EC (voir ci-dessous).

Le numéro de sprite donne la forme à utiliser à l'affichage. On peut définir 256 sprites 8 sur 8, ou 64 sprites 16 sur 16. pour les sprites 8 sur 8, on choisit simplement le numéro de celui que l'on désire afficher. Pour les sprites 16 sur 16, on donne le numéro de sprite multiplié par quatre.

La couleur est toujours codée sur 4 bits.

Le bit EC (« Early Clock ») sert à décaler le sprite de 32 pixels vers la gauche. Ceci permet de faire « entrer » un sprite à l'écran comme dans l'exemple suivant :

```
10 screen 2 : color 10, 0, 0 : cls
20 sprite$(1) = string$(8, 255)
30 for i = 24 to 40 : for j = 0 to 100 : next j
40 put sprite 0, (i, 35) : vpoke &h1b03, &h87
50 next i
55 if not (strig(0)) then 55
```

Le mode 2 (MSX2)

Fonctionnement :

Dans ce mode, il peut y avoir 32 sprites à l'écran numérotés de 0 à 31. Plus un sprite a un numéro faible, plus il est prioritaire. Ceci signifie que si le sprite n°3 croise le sprite n°16, on verra le sprite n°3 passer dessus (lorsque des pixels des deux sprites se trouveront aux mêmes coordonnées, c'est les pixels du sprite n°3 qui seront affichés).

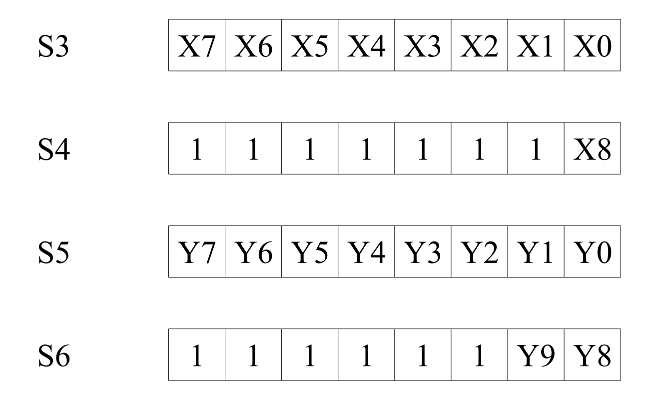
8 sprites peuvent, au maximum, se trouver sur la même ligne. A partir du 9ème sprite, toutes les portions communes aux 9 sprites disparaissent sur le (ou les) sprites de plus faible priorité.

Paramètres :

Matrice des sprites -	bit 1 (SI) du registre 1 du VDP
	SI à 0 = sprites 8 sur 8
	SI à 1 = sprites 16 sur 16
Taille des sprites -	bit 0 (MAG) du registre 1 du VDP
	MAG à 0 = taille normale
	MAG à 1 = taille double (16 sur 16, ou 32 sur 32, avec de gros pixels)
Etat des sprites -	bit 1 (SPD) du registre 8 du VDP
	SPD à 0 = sprites affichés (enable)
	SPD à 1 = sprites éteints (disable)

Collisions :

Lorsque deux sprites entrent en collision (deux pixels ou plus ont les mêmes coordonnées), le bit 5 du registre d'état (status) n°0 passe à 1 (si le bit CC est à 0). Le bit 5 est automatiquement remis à 0 lors d'une lecture du registre d'état n°2. Les registres d'état n°3, 4, 5 et 6 indiquent alors les coordonnées du pixel où la collision s'est produite (à condition que les bits MS et LP du registre n°8 du processeur soient tous les deux à 0).



S3	X7	X6	X5	X4	X3	X2	X1	X0

S4	1	1	1	1	1	1	1	X8

S5	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0

S6	1	1	1	1	1	1	Y9	Y8

Lorsque le registre n°5 est lu, les registres sont tous les 4 remis à 0.

Une possibilité supplémentaire est offerte au programmeur dans le mode 2. En effet, si le bit CC est mis à 1 dans la table des couleurs de sprite, alors un « ou » logique (OR) est effectué entre les deux lignes de sprites superposées.

<u>9 sprites</u> :

Lorsque plus de 8 sprites se trouvent sur la même ligne, le bit 6 (5S) du registre d'état (status) n°0 passe à 1. De plus les 5 bits de poids faible du registre d'état n°0 donnent le numéro du 9ème sprite (0 à 31).

<u>Utilisation</u> :

Pour afficher un sprite, il faut d'abord le définir dans la table des formes de sprites.

Pour les sprites 8 sur 8, il suffit de 8 octets pour définir un sprite, sachant qu'un bit représente un pixel, par exemple, en SCREEN 7, la table des formes de sprites pourrait être :

	7 0	huit octets pour le sprite n°0 qui aura cette forme
0F000H	11111111	X X X X X X X X
0F001H	11000011	X X . . . . X X
0F002H	10100101	X . X . . X . X
0F003H	10011001	X . . X X . . X
0F004H	10011001	X . . X X . . X
0F005H	10100101	X . X . . X . X
0F006H	11000011	X X . . . . X X
0F007H	11111111	X X X X X X X X

Dans le cas de sprites 16 sur 16, les choses deviennent un tout petit peu plus compliquées. Le sprite se décompose en 4 parties :

1	3
2	4

	7 0									
0F000H	11111110	X	X	X	X	X	X	X		
0F001H	10000010	X						X		
0F002H	10001110	X				X	X	X		
0F003H	10001000	X				X				
0F004H	10001000	X				X				huit octets pour le premier quart du sprite n°0
0F005H	10001000	X				X				
0F006H	10001000	X				X				
0F007H	10001000	X				X				
0F008H	10001000	X				X				
0F009H	10001000	X				X				
0F00AH	10001000	X				X				
0F00BH	10001000	X				X				huit octets pour le second quart du sprite n°0
0F00CH	10001000	X				X				
0F00DH	10001111	X				X	X	X	X	
0F00EH	10000000	X								
0F00FH	11111111	X	X	X	X	X	X	X	X	
0F010H	01111111		X	X	X	X	X	X	X	
0F011H	01000001		X						X	
0F012H	01110001		X	X	X				X	
0F013H	00010001				X				X	huit octets pour le troisième quart du sprite n°0
0F014H	00010001				X				X	
0F015H	00010001				X				X	
0F016H	00010001				X				X	
0F017H	00010001				X				X	
0F018H	00010001				X				X	
0F019H	00010001				X				X	
0F01AH	00010001				X				X	
0F01BH	00010001				X				X	huit octets pour le dernier quart du sprite n°0
0F01CH	00010001				X				X	
0F01DH	11110001	X	X	X	X				X	
0F01EH	00000001								X	
0F01FH	11111111	X	X	X	X	X	X	X	X	

La réunion des quatre parties donne le résultat ci-dessous :

1	3
2	4

```
X X X X X X X     X X X X X X X
X           X     X           X
X       X X X     X X X       X
X       X             X       X
X       X             X       X
X       X             X       X
X       X             X       X
X       X             X       X
X       X             X       X
X       X             X       X
X       X             X       X
X       X             X       X
X       X             X       X
X       X X X X X X X X       X
X                             X
X X X X X X X X X X X X X X X X
```

Il faut ensuite remplir la table des couleurs de sprites :

	7	6	5	4	3 2 1 0	
0F800H	EC	CC	IC	0	couleur ligne 0	couleur de chaque ligne d'un sprite 8 sur 8 ou 16 sur 16 numéro 0
0F801H	EC	CC	IC	0	couleur ligne 1	
0F802H	EC	CC	IC	0	couleur ligne 2	
0F803H	EC	CC	IC	0	couleur ligne 3	
...					...	
0F80EH	EC	CC	IC	0	couleur ligne 14	
0F80FH	EC	CC	IC	0	couleur ligne 15	
0F810H	EC	CC	IC	0	couleur ligne 0	idem numéro 1
0F811H	EC	CC	IC	0	couleur ligne 1	

Le bit EC (« Early Clock ») sert à décaler le sprite de 32 pixels vers la gauche. Ceci permet de faire « entrer » un sprite à l'écran.

Le bit CC permet de choisir le mode d'affichage des sprites lors d'une collision :

CC à 0 : le sprite prioritaire est affiché

CC à 1 : un « ou » logique (OR) est effectué sur les portions de sprite entrés en collision.

Le bit IC sert à avtiver la détection de collision :

IC à 0 : détection activée

IC à 1 : détection interdite

La couleur est toujours codée sur 4 bits, même en mode d'écran 6 (SCREEN 6) bien que l'on ne dispose que de 4 couleurs (voir le mode d'écran 6 pour plus de précisions, fonction « hardware tiling »).

En SCREEN 8, les valeurs sont fixes et non redéfinissables. En voici le détail :

couleur	rouge			vert			bleu		
	R2	R1	R0	G2	G1	G0	B2	B1	B0
0000B	0	0	0	0	0	0	0	0	0
0001B	0	0	0	0	0	0	0	1	0
0010B	0	0	0	0	1	1	0	0	0
0011B	0	0	0	0	1	1	0	1	0
0100B	0	1	1	0	0	0	0	0	0
0101B	0	1	1	0	0	0	0	1	0
0110B	0	1	1	0	1	1	0	0	0
0111B	0	1	1	0	1	1	0	1	0
1000B	1	0	0	1	1	1	0	1	0
1001B	0	0	0	0	0	0	1	1	1
1010B	0	0	0	1	1	1	0	0	0
1011B	0	0	0	1	1	1	1	1	1
1100B	1	1	1	0	0	0	0	0	0
1101B	1	1	1	0	0	0	1	1	1
1110B	1	1	1	1	1	1	0	0	0
1111B	1	1	1	1	1	1	1	1	1

Notez que vous pouvez régler la couleur, aussi que les autres bits, pour CHAQUE ligne du sprite. Notez également qu'on utilise toujours 16 octets, même pour les sprites 8 sur 8.

Une fois le sprite défini, on peut le manipuler à souhait grâce à la table des attributs de sprites.

Cette table fonctionne avec des plans graphiques, et compte 4 octets par plan. On appelle « plan graphique » un espace comprenant un élément graphique. Il y a donc 32 plans (numérotés de 0 à 31) avec chacun un sprite, puis un plan avec les graphismes ordinaires, enfin parfois un plan en entrée vidéo.

La table des attributs de sprites code les caractéristiques de chaque plan sur 4 octets :

	7 6 5 4 3 2 1 0	
01B00H	ordonnée sprite plan 0 (0-255)	attributs du plan 0
01B01H	abscisse sprite plan 0 (0-255)	
01B02H	numéro de sprite (0-255)	
01B03H	---- réservé au système ----	
01B04H	ordonnée sprite plan 1 (0-255)	

L'ordonnée peut varier entre 0 et 255. Si le sprite sort de l'écran, le V9938 n'affichera que la partie visible du sprite. En réalité, on utilise les coordonnées de 225 (-31 en complément à 2) à 255, et 0 à 191. Ceci permet de faire entrer un sprite dans l'écran (-31 à -1 ou 225 à 255) ou au contraire de le faire sortir (183 à 191 pour un 8 sur 8) petit à petit.

Notez que si l'ordonnée de n'importe quel sprite est à 216 (0D8H), alors tous les sprites dans les plans supérieurs deviennent invisibles. Par exemple si l'on met l'ordonnée du sprite contenu dans 8 à 216, alors tous les sprites dans les plans 9 à 31 disparaissent.

L'abscisse prend une valeur entre 0 et 255. Dans les modes à 512 pixels de large (SCREEN 6 et 7), un pixel sprite équivaut à deux pixels graphiques. Pour faire sortir le sprite on utilise les abscisses entre 247 et 255 (pour un sprite 8 sur 8). Par contre, pour le faire entrer, il faut utiliser le bit EC dans la table des couleurs.

Le numéro de sprite donne la forme à utiliser à l'affichage. On peut définir 256 sprites 8 sur 8, ou 64 sprites 16 sur 16. pour les sprites 8 sur 8, on choisit simplement le numéro de celui que l'on désire afficher. Pour les sprites 16 sur 16, on donne le numéro de sprite multiplié par quatre.

5.11 LA SOURIS ET LE CRAYON OPTIQUE

Ces deux périphériques de pointage peuvent être gérés directement par le processeur vidéo V9938. Il est cependant préférable d'utiliser le Bios (voir le chapitre 3, routine NEWPAD en Sub-ROM) chaque fois que cela est possible.

Néanmoins, pour ceux qui ont un besoin absolu de vitesse d'exécution, voici la marche à suivre pour un accès direct par le processeur vidéo.

LA SOURIS

– Il faut d'abord régler les 2 bits de poids fort du registre 8 du processeur vidéo :

R8	MS	LP	TP	CB	VR	0	SPD	BW

Mettre MS (MouSe enable) à 1 et LP à 0.

– A partir de ce moment, on peut lire le registre d'état (status) n°1 :

S1	FL	LPS						FH

FL : 0 = le bouton 1 de la souris n'a pas été enfoncé
1 = le bouton 1 de la souris a été enfoncé

LPS : 0 = le bouton 2 de la souris n'a pas été enfoncé
1 = le bouton 2 de la souris a été enfoncé

Notez que ces bits sont automatiquement remis à 0 lors d'une lecture du registre d'état n°2.

– le déplacement relatif de la souris peut alors être lu dans les registres d'état 3 et 5 du VDP :

S3	X7	X6	X5	X4	X3	X2	X1	X0
S5	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0

Les valeurs de déplacement sont données en complément à 2. Dès que le registre n°15 (registre d'accès aux registres d'état) est chargé avec 3 ou 5, les valeurs de déplacement n'évoluent plus. Il ne faut pas négliger de changer la valeur du registre 15 pour recommencer à tester la souris.

LE CRAYON OPTIQUE

– Il faut d'abord régler les 2 bits de poids fort du registre 8 du processeur vidéo :

R8	MS	LP	TP	CB	VR	0	SPD	BW

Mettre LP (Light Pen enable) à 1 et MS à 0. Si vous désirez qu'une interruption se produise lorsque le crayon optique détecte de la lumière, mettre le bit 5 du registre 0 à 1. Ce bit est automatiquement remis à 0 lors d'une lecture du registre d'état n°1 du processeur vidéo.

– A partir de ce moment, on peut lire le registre d'état (status) n°1 :

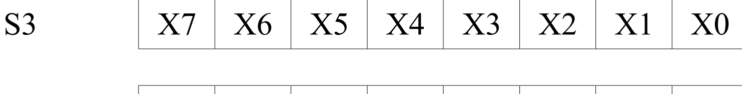


S1	FL	LPS						FH

FL : 0 = aucune lumière n'a été détectée
1 = de la lumière a été détectée

LPS : 0 = le bouton du crayon optique n'a pas été enfoncé
1 = le bouton du crayon optique a été enfoncé

– les coordonnées du pixel où la lumière a été détectée se trouvent alors dans les registres d'état 3, 4, 5 et 6 du processeur vidéo



S3	X7	X6	X5	X4	X3	X2	X1	X0
S4	0	0	0	0	0	0	0	X8
S5	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
S6	0	0	0	0	0	0	Y9	Y8

Les valeurs dans ces registres demeurent inchangées tant que le registre d'état n°5 n'a pas été lu. Il

faut donc lire les registres 3, 4, 6 puis seulement 5.

6 DES APPLICATIONS TYPES

Ce chapitre est destiné à soulager le programmeur en lui proposant des solutions « prêtes à l'emploi » aux problèmes qu'on rencontre dans la majorité des applications. Muni de cette « bibliothèque de base », le programmeur ne perdra plus de temps en recherches inutiles et pourra se consacrer entièrement à la programmation de son application.

6.1 REVENIR A L'INTERPRETEUR BASIC

Il est souvent utile de pouvoir retourner à l'interpréteur Basaic depuis un programme en langage machine, autrement que par le RET du Z80. Pour cela il faut :

1 - sélectionner la Main-ROM en pages 0 et 1.

2 - effectuer un saut en 0409BH.

Ce qui donnerait, en langage machine, les lignes de programme suivantes :

```
            ORG   adrs
;
MNROM       EQU   0FCC1H
ENASLT      EQU   00024H
RETURN      EQU   0409BH
;
            ........
;
            LD    A, (MNROM)
            LD    HL, 0              ; Bios en page 0
            CALL  ENASLT
            LD    HL, 04000H         ; Basic en page 1
            CALL  ENASLT
            JP    RETURN
```

Lorsque l'on travaille sous Basic, ou que l'on ne touche pas aux slots, il suffit d'effectuer le JP 0409BH.

6.2 MSX1 OU MSX2 ?

Pour savoir si l'ordinateur utilisé est un MSX1 ou un MSX2, il suffit d'examiner le contenu de la case mémoire 0002DH en Main-ROM.

contenu de 02DH	ordinateur
0	MSX1

1	MSX2
2	MSX3
3 à 255	non défini

Attention : la Main-ROM ne se trouve pas toujours en slot 0-0 sur MSX2. Il faut lire le contenu de la variable système MN-ROM pour connaître le slot de la Main-ROM.

6.3 LE « PRINT » EN LANGAGE MACHINE

Je vous propose une petite routine - très simple - fort utile en mode texte. Elle permet d'afficher une chaîne de caractères à l'écran en 32, 40 ou 80 colonnes :

```
ECRIRE :    LD      A, (HL)
            CP      0
            RET     Z
            CALL    0A2H
            INC     HL
            JR      ECRIRE
```

Il suffit de charger dans HL l'adresse du premier caractère de la chaîne à afficher. Puis d'appeler la routine, comme dans l'exemple suivant :

```
EXEMPL :    LD      HL, DATA
            CALL    ECRIRE
            RET
DATA :      DB      'COUCOU', 0
;
```

Ne pas oublier de mettre un 0 à la fin de la chaîne. Cette routine comme le PRINT du Basic gère les codes CTRL et ESC.

6.4 SYSTEME A DISQUETTES OU A CASSETTES ?

Pour savoir si l'ordinateur possède un lecteur de disquettes ou non, il faut vérifier l'état d'un hook modifié par le système d'exploitation de disquette. Le hook H.PHYD convient parfaitement. Ainsi, si 0FFA7H - adresse de H.PHYD - contient 0C9H (code du RET en Z80), vous travaillez sur un système à cassette. Si 0FFA7H contient autre chose que 0C9H (qui devrait être 0F7H, code du RST 30), alors l'ordinateur possède un lecteur de disquette, intégré ou non.

De plus, lorsqu'un lecteur de disquette est présent, certaines variables système sont modifiées :

adresse	contenu
0FB21H	nombre de lecteurs sur le premier contrôleur
0FB22H	numéro de slot du premier contrôleur
0FB23H	nombre de lecteurs sur le second contrôleur

0FB24H	numéro de slot du second contrôleur
0FB25H	nombre de lecteurs sur le troisième contrôleur
0FB26H	numéro de slot du troisième contrôleur
0FB27H	nombre de lecteurs sur le quatrième contrôleur
0FB28H	numéro de slot du quatrième contrôleur

Un MSX peut donc gérer jusqu'à 8 lecteurs de disquettes simultanément. S'il y a moins de quatre contrôleurs, les variables système correspondant aux contrôleurs inexistants contiennent 0.

En cas d'absence de lecteur de disquette, cette zone mémoire n'est pas initialisée et contient donc n'importe quoi.

6.5 TRAITER LES ERREURS LIEES AUX DISQUETTES

Il est fort désagréable de voir s'afficher, au milieu d'un programme en français, un message du genre « Disk write protected » ou « Disk offline ». Il est heureusement possible de remédier à ce genre de désagrément. Voici la marche à suivre :

1 - modifier l'adresse contenue en 0F323H et 0F324H pour détourner le traitement des erreurs disque vers votre propre routine.

2 - votre routine pourra envoyer les messages adéquats sachant que l'accumulateur contient le numéro du drive, alors que le bit 0 du registre C est un flag indiquant si l'erreur s'est produite lors d'une lecture (bit à 0) ou d'une écriture (bit à 1). Les bits 1, 2 et 3 indiquent le type d'erreur suivant la table :

3 2 1	type d'erreur
0 0 0	protégée contre l'écriture (« Write protected »)
0 0 1	pas prêt (« Disk offline »)
0 1 0	erreur de CRC (« CRC error »)
0 1 1	erreur de lecture (« Seek error »)
1 0 0	fichier non trouvé (« File not found »)
1 0 1	erreur d'écriture (« Write error »)
1 1 0	autre erreur

3 - Votre routine doit rendre le contrôle au DOS par un RET après avoir chargé le registre C avec l'action à exécuter (0 = Ignore, 1 = Retry, 2 = Abort).

6.6 FAIRE DE LA MUSIQUE EN LANGAGE MACHINE

La méthode classique pour faire de la musique en langage machine consiste à utiliser une interruption pour charger les registres du PSG 8910 à intervalle régulier.

Le programmeur peut aussi utiliser le système des queues musicales. Malheureusement, cette méthode est difficile à mettre en œuvre.

Il existe une méthode très simple pour exécuter en langage machine l'équivalent de l'instruction PLAY du Basic. Mais cette méthode N'EST PAS GARANTIE officiellement par Microsoft et ASCII. Quelque peu empirique, elle est cependant officieusement garantie, puisqu'ASCII a affirmé qu'elle ne serait pas modifiée. Cela se vérifie pour l'instant puisque tous les MSX vendus en France à ce jour exécutent parfaitement cette routine. Vous utilisez cette routine à vos risques et périls quant à une éventuelle compatibilité avec les MSX à venir.

Pour exécuter cette routine, il faut charger le registre double HL avec l'adresse du premier octet de la chaîne de caractères à jouer (la chaîne est la même qui pour un PLAY en Basic), puis appeler la routine située à l'adresse 073E5H en Main-ROM. Voici un exemple en langage machine :

```
                ORG   0C000H
;
PLAY            EQU   073E5H
;
                LD    HL, DATA
                CALL  PLAY
                RET
;
DATA:           DB    022H
                DB    'O5L6DABDACF'
                DB    ' « , »'
                DB    'O3L4DEDE'
                DB    ' « , »'
                DB    'O7L8ADEFGED'
                DB    022H, 000H
```

Attention à ne pas oublier le 0 en fin de chaîne. Dans l'exemple ci-dessus, on présuppose que la Main-ROM est sélectionnée.

6.7 PASSAGE DE PARAMETRES BASIC/LANGAGE MACHINE

Dans la plupart des applications, il n'est pas indispensable que l'intégralité du programme soit écrit en langage machine. Il suffit bien souvent de quelques routines en Z80 pour donner un look professionnel à un programme en Basic. Dans ces cas, le problème de l'échange d'informations entre Basic et langage machine se pose.

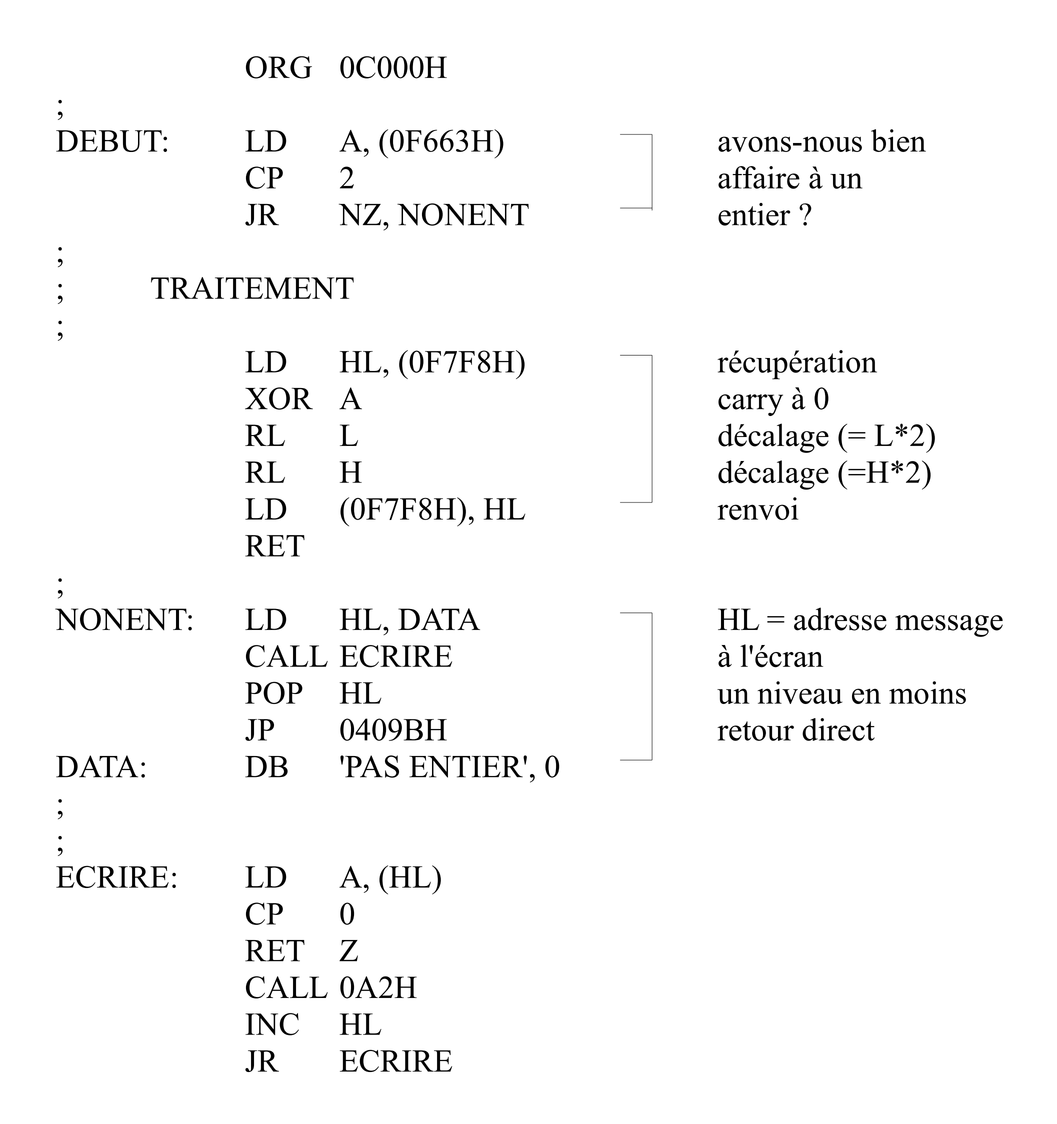
La méthode classique consiste à définir une zone de communication à laquelle on accède sous Basic par les instructions POKE et PEEK. Cette méthode présente plusieurs inconvénients : l'exécution est plutôt lente, les instructions sont longues et occupent un précieux espace mémoire, toutes les données sont enregistrées deux fois (donc perte très importante de place mémoire).

La fonction USR du Basic (que la majorité des utilisateurs - y compris moi - emploie comme une simple instruction CALL) permet l'échange automatique de données dans les 2 sens. Voyons dans le détail le fonctionnement de l'instruction USR. Plusieurs cas se présentent suivant la nature du paramètre à passer :

<u>Le paramètre est un entier</u> :

C'est le cas le plus simple. La fonction USR met 2 dans la case mémoire 0F663H pour indiquer un entier. La valeur du paramètre se trouve sur 2 octets à l'adresse 0F7F8H. Après traitement, 0F663H doit contenir 2 si l'on renvoie un entier au Basic. Ce dernier se trouvera en 0F7F8H et 0F7F9H.

Prenons un exemple : nous désirons réaliser un programme qui multipliera automatiquement un entier positif par 2 :

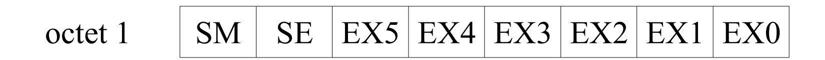


```
            ORG   0C000H
;
DEBUT:      LD    A, (0F663H)        avons-nous bien
            CP    2                  affaire à un
            JR    NZ, NONENT         entier ?
;
;       TRAITEMENT
;
            LD    HL, (0F7F8H)       récupération
            XOR   A                  carry à 0
            RL    L                  décalage (= L*2)
            RL    H                  décalage (=H*2)
            LD    (0F7F8H), HL       renvoi
            RET
;
NONENT:     LD    HL, DATA           HL = adresse message
            CALL  ECRIRE             à l'écran
            POP   HL                 un niveau en moins
            JP    0409BH             retour direct
DATA:       DB    'PAS ENTIER', 0
;
;
ECRIRE:     LD    A, (HL)
            CP    0
            RET   Z
            CALL  0A2H
            INC   HL
            JR    ECRIRE
```

<u>Le paramètre est en simple précision</u> :

C'est à peine plus compliqué. La fonction USR met 4 dans la case mémoire 0F663H pour indiquer la simple précision. La valeur du paramètre se trouve sur 4 octets à partir de l'adresse 0F7F6H. Après traitement, 0F663H doit contenir 4 si l'on renvoie au Basic un nombre en simple précision. Ce dernier se trouvera en 0F7F6H et 0F7F9H.

Les 4 octets codent le nombre en simple précision de la manière suivante :



octet 1	SM	SE	EX5	EX4	EX3	EX2	EX1	EX0

SM : signe de la mantisse 0 = positif, 1 = négatif

SE : signe de l'exposant 0 = négatif, 1 = positif

EX0 à EX5 : valeur de l'exposant (0-31)

octet 2	PC3	PC2	PC1	PC0	SC3	SC2	SC1	SC0

PC0 à PC3 : premier chiffre en BCD

SC0 à SC3 : second chiffre en BCD

octet 3	TC3	TC2	TC1	TC0	QC3	QC2	QC1	QC0

TC0 à TC3 : troisième chiffre en BCD

QC0 à QC3 : quatrième chiffre en BCD

octet 4	CC3	CC2	CC1	CC0	SC3	SC2	SC1	SC0

CC0 à CC3 : cinquième chiffre en BCD

SC0 à SC3 : sixième chiffre en BCD

Par exemple - 3483200 serait codé :

-3483200 = -0,348320 * 10^7

donc

le premier octet contient 11000111B, soit 0C7H

le second octet code les 2 premiers chiffres, soit 034H

le troisième octet code les 2 chiffres suivants, soit 083H

le quatrième octet code les deux derniers chiffres, soit 020H

<u>Le paramètre est en double précision</u> :

la fonction USR met 8 dans la case mémoire 0F663H pour indiquer la double précision. la valeur du paramètre se trouve sur 8 octets à partir de l'adresse 0F7F6H. après traitement, 0F663H doit contenir 8 si l'on renvoie au Basic un nombre en double précision. Ce dernier se trouvera en 0F7F6H et 0F7FDH.

Les 8 octets codent le nombre en double précision de la même manière qu'en simple précision (voir ci-dessus pour plus de précisions), si ce n'est que la mantisse est sur 7 octets au lieu de 3.

<u>Le paramètre est une chaîne de caractères</u> :

La fonction USR met 3 dans la case mémoire 0F663H pour indiquer une chaîne de caractères. Elle charge ensuite l'adresse du « string descriptor » en 0F7F8H et 0F7F9H. Après traitement, 0F663H

doit contenir 3 si l'on renvoie au Basic une chaîne de caractères. L'adresse du nouveau « string descriptor » doit être en 0F7F8H et 0F7F9H.

Le « string descriptor » est une zone mémoire de 3 octets qui, pour une chaîne de caractères, contient la longueur de la chaîne sur le premier octet, puis l'adresse du premier octet de la chaîne sur les 2 octets suivants.

Voici un exemple qui transforme tous les caractères de majuscules en minuscules dans une chaîne :

```
                ORG   0C000H
;
                LD    A, (0F663H)
                CP    3
                RET   NZ
;
                LD    HL, (0F7F8H)
                LD    B, (HL)
                INC   HL
                LD    E, (HL)
                INC   HL
                LD    D, (HL)
                EX    DE, HL
LOOP :                LD    A, (HL)
                OR    020H
                CALL  0A2H
                DJNZ  LOOP
                RET
```

Après avoir tapé et assemblé ce programme, entrez les lignes Basic suivantes et faites RUN :

```
10 CLS
20 CLEAR 200, &HC000
30 DEFUSR = &HC000
40 A$ = « TRANSFORMATION »
50 A$ = USR(A$)
60 PRINT : PRINT : LIST
```

6.8 LES CODES DE CONTROLE [CTRL]

Chacun sait que sous Basic, en enfonçant simultanément les touches « CTRL » et « L », on efface l'écran. Le MSX dispose de plusieurs autres codes du même type qui peuvent être utilisés en mode direct ou dans un programme. En voici la liste :

code	touches	effet
2	CTRL + B	place le curseur au début du mot précédent

3	CTRL + C	interrompt le programme (BREAK)
5	CTRL + E	efface toute la ligne à droite du curseur
6	CTRL + F	place le curseur au début du mot suivant
7	CTRL + G	émission d'un bref bip sonore
8	CTRL + H	back space (BS)
9	CTRL + I	tabulation / idem touche TAB
10	CTRL + J	descend le curseur d'une ligne
11	CTRL + K	idem touche EFE ou HOME
12	CTRL + L	effacement de l'écran
13	CTRL + M	effectue un RETURN / idem touche RETURN
14	CTRL + N	place le curseur en fin de ligne
18	CTRL + R	idem touche INS
21	CTRL + U	efface la ligne ou se trouve le curseur
24	CTRL + X	idem touche SELECT
27	CTRL + [	idem touche ESC
28	CTRL + \	idem flèche droite
29	CTRL +]	idem flèche gauche
30	CTRL + ^	idem flèche haut
31	CTRL + _	idem flèche bas

Essayez donc cet exemple (MSX2) :

```
10 SCREEN 0 : WIDTH 80 : COLOR 10, 0, 0 : CLS
20 A$ = « Ce texte s'affiche bizarrement !!! »
25 LOCATE 25, 10
30 FOR I = 1 TO LEN (A$)
40 PRINT CHR$(30) + MID$(A$, I, 1);
50 NEXT I
```

6.9 LES CODES ESCAPE [ESC]

Le MSX peut utiliser toute la série des codes ESC compatible avec les terminaux VT-52 ou HEATH-19 dont la liste suit :

code	effet
ESC A - 27 65	monte le curseur d'une ligne
ESC B - 27 66	descend le curseur d'une ligne

ESC C - 27 67	déplace le curseur vers la droite
ESC D - 27 68	déplace le curseur vers la gauche
ESC H - 27 72	curseur en haut à droite (HOME)
ESC Y col ligne	curseur en X, Y (LOCATE) voir ci-dessous
ESC j - 27 106	efface l'écran (CLS)
ESC E - 27 69	efface l'écran (CLS)
ESC K - 27 75	efface jusqu'à la fin de la ligne
ESC J - 27 74	efface jusqu'à la fin de l'écran
ESC l - 27 108	efface toute la ligne
ESC L - 27 76	insère une ligne
ESC M - 27 77	détruit une ligne
ESC x4 - 27 120 52	définit un curseur de type entier
ESC x5 - 27 120 53	éteint le curseur
ESC y4 - 27 121 52	définit un curseur de type barre
ESC y5 - 27 121 53	allume le curseur

La séquence ESC Y est légèrement plus compliquée à utiliser. Il faut en effet envoyer les deux octets de ESC Y (27 et 89), puis les faire suivre par deux octets qui définissent la colonne (n° de colonne + 020H) ainsi que la ligne (n°de la ligne + 020H). Affichons par exemple la chaîne « ICI » à la position (10, 33) :

10 PRINT CHR$(27) + « Y*AICI »

On envoie un CHR$(27) puis « Y » suivi de « * » (code ASCII = 42, soit 10 + 020H), et « A » code ASCII = 65 soit 33 + 020H).

Très peu de gens connaissent l'existence de ces codes ESC qui peuvent pourtant parfois rendre de grands services comme le démontre l'exemple suivant :

```
10 SCREEN 0 : WIDTH 40 : COLOR 10, 0, 0 : CLS : KEYOFF
20 E$ = CHR$(27)
30 FOR I = 0 TO 17 : PRINT « LIGNE »I : NEXT
40 PRINT : PRINT « On peut effacer les dernières lignes. » : PRINT
50 PRINT « Comme dans une fenêtre... » : PRINT
60 PRINT TAB(5) »... aussi rapidement qu'avec un CLS »;
70 FOR I = 1 TO 1500 : NEXT
80 LOCATE 0, 19 : PRINT « Sans toucher aux autres ! »+E$ + « J »
90 FOR I = 1 TO 1500 : NEXT I : LOCATE 0,18 : GOTO 40
```

Essayez d'enlever le + E$ + « J » à la ligne 80.

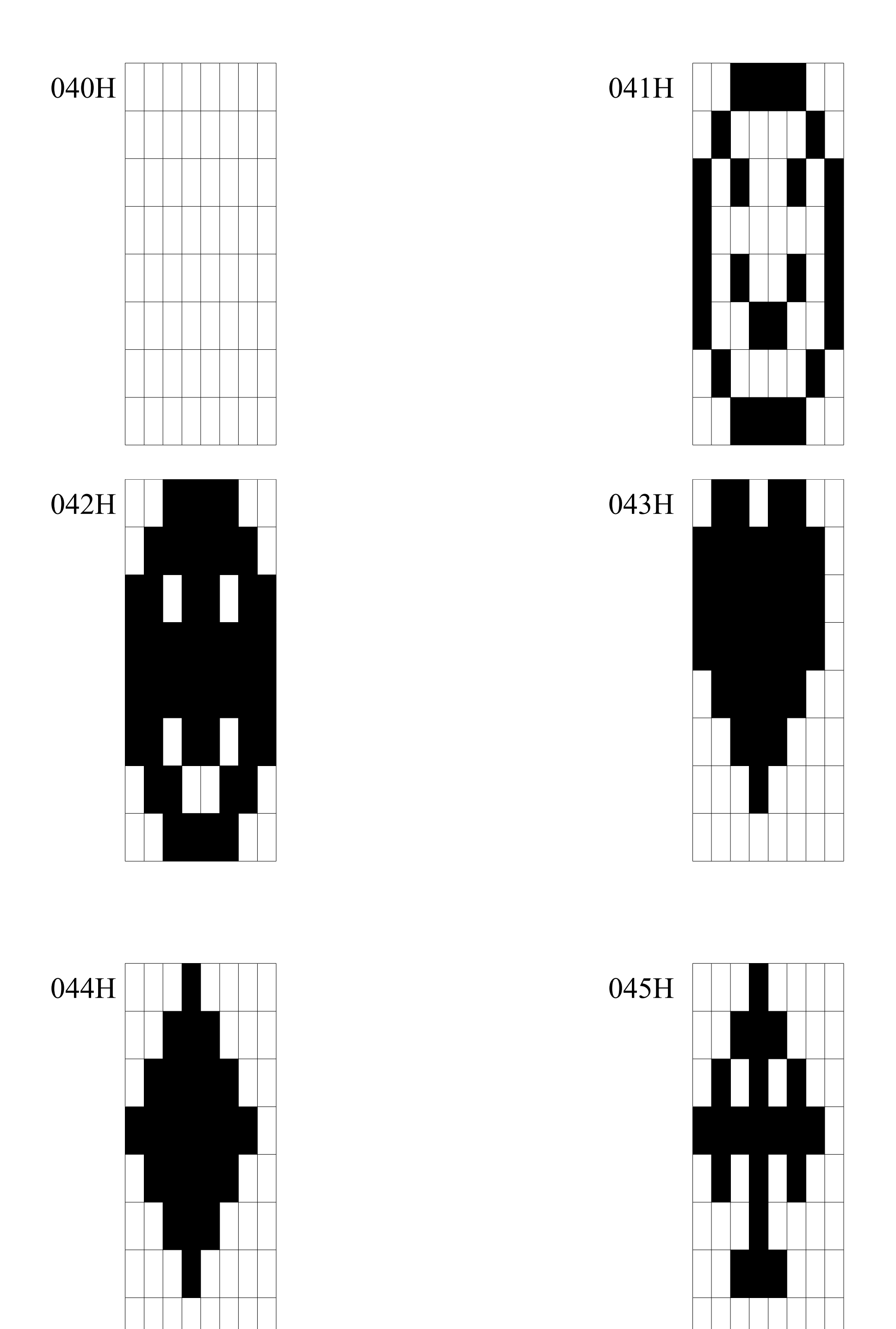
6.10 UTILISER LE SECOND JEU DE CARACTERES

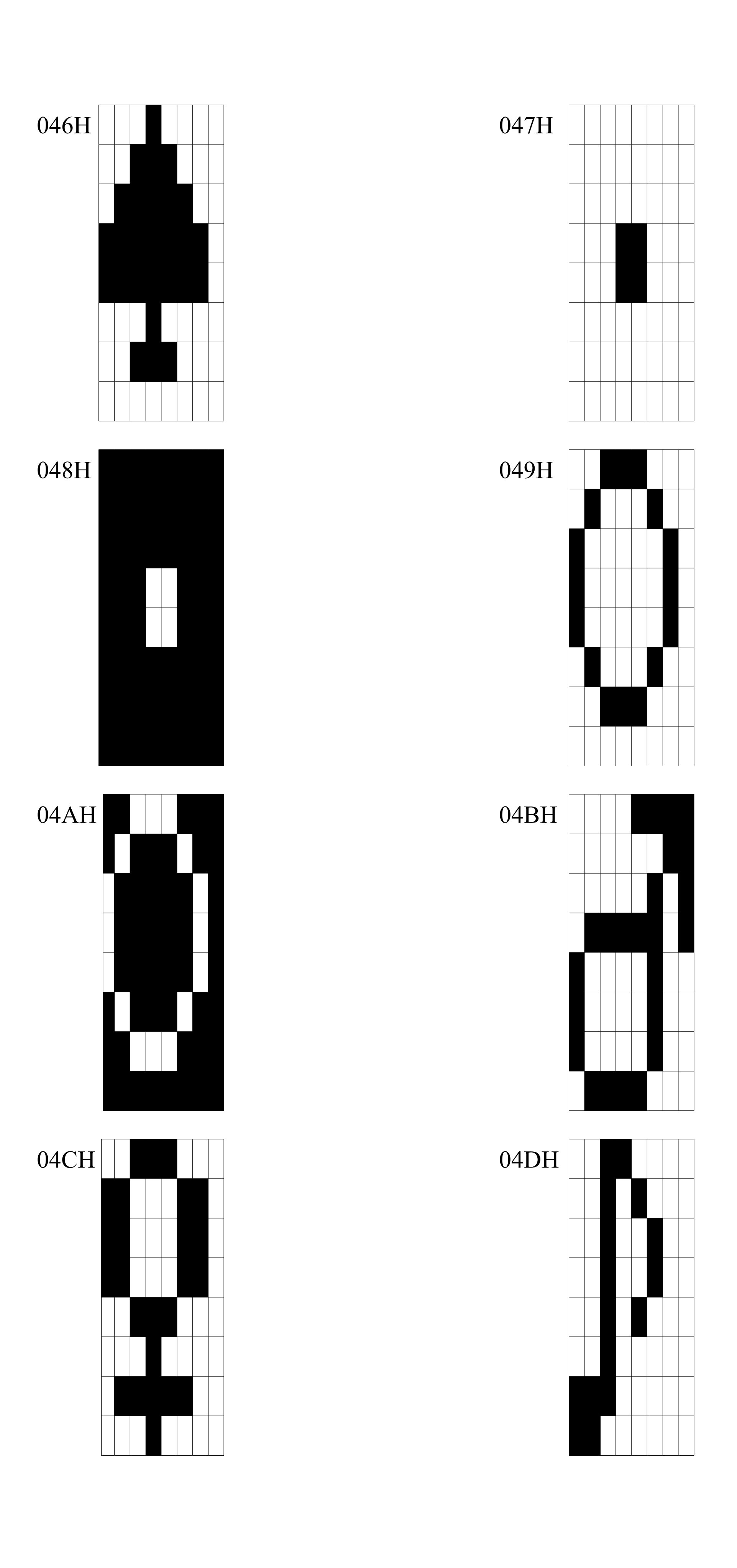
Les MSX possèdent un second jeu de caractères. Ce dernier est réduit (32 caractères) et on y accède par un CHR$(1) suivi d'un code ASCII entre 64 (040H) et 95 (05FH). Par exemple, en SCREEN 1, la ligne :

PRINT CHR$(1) + « A »

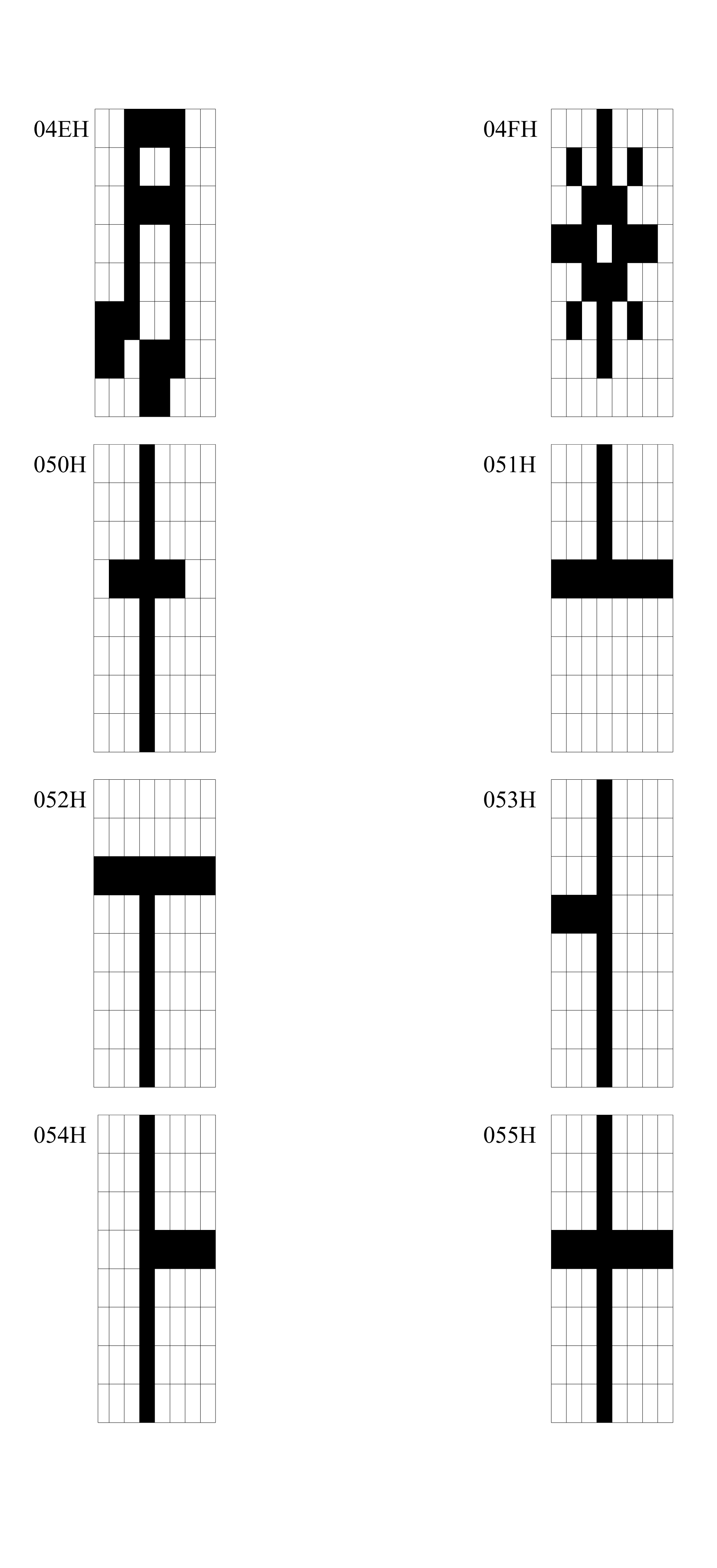
fera s'afficher à l'écran une petite tête sympathique. Notez que tous ces caractères se trouvent définis dans des matrices 8 sur 8. On perdra donc, en SCREEN 0, les 2 bits de poids faible, et le caractère sera amputé de ses 2 colonnes les plus à droite.

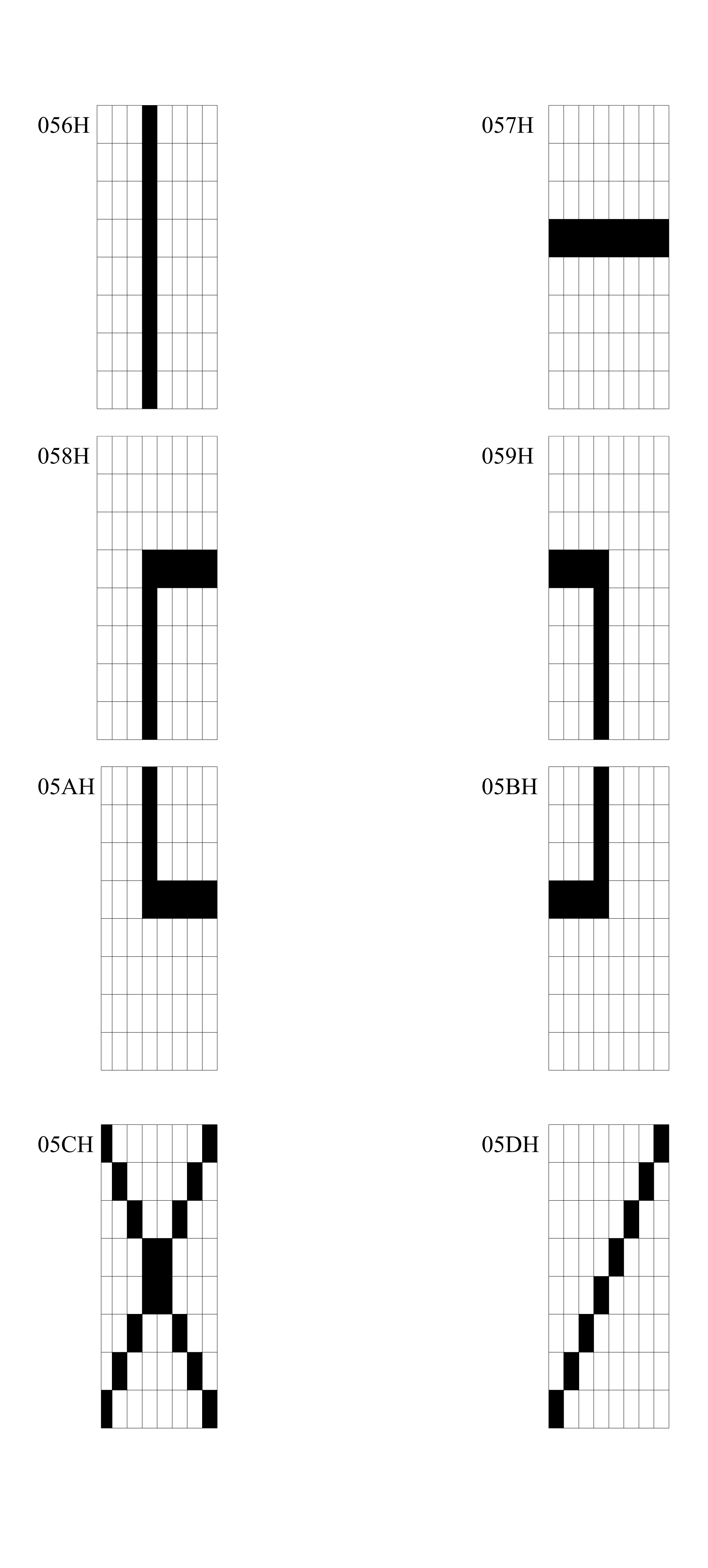
Voici les matrices des caractères de ce second jeu :



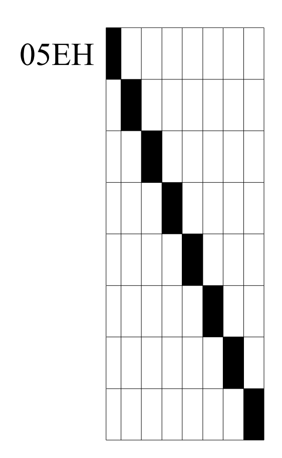


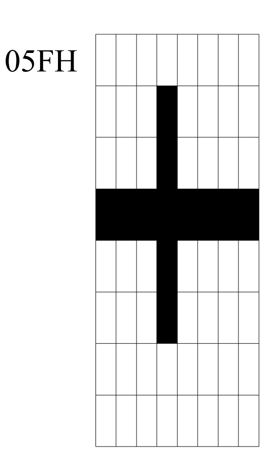
046H
047H
048H
049H
04AH
04BH
04CH
04DH


04EH
04FH
050H
051H
052H
053H
054H
055H



056H
057H
058H
059H
05AH
05BH
05CH
05DH

05EH 

05FH 

6.11 TROUVER DE LA RAM EN PAGES 0 ET 1

Un problème incontournable sur les MSX (dès que l'on désire utiliser plus de 32 Ko de mémoire vive) est la recherche et l'utilisation de mémoire vive aux adresses 0 à 0800H.

La recherche :

Le court programme suivant détermine dans quel slot se trouve de la mémoire vive pour la page 1 (04000H). Si vous désirez effectuer une recherche pour la page 0, il faut remplacer le LD HL, 04000H (sixième ligne) par LD HL, 0.

A la fin de cette routine, HL contient toujours le numéro de la page et l'accumulateur contient le numéro de slot sous la forme :

F000SSPP en binaire.

sachant que :

F	est un flag qui indique le type de slot (0 = primaire , 1 = secondaire)
SS	donne le numéro de slot secondaire (0-3)
PP	donne le numéro de slot primaire (0-3)

Si votre système possède moins de 64 Ko, le programme sortira par la routine « RIEN: », dans le cas contraire, il passera par la routine « RAM: ». Le sous-programme « RETOUR: » rend la main en sélectionnant le Basic.

```
            ORG   0C000H
;
ENASLT      EQU   00024H
;
FIND:       LD    B, 0FH
            LD    HL, 04000H
LOOP:       LD    A, B
            OR    080H
            PUSH  BC
```

```
                PUSH  AF
                PUSH  HL
                CALL  ENASLT
                POP   HL
                POP   AF
                LD    (HL), A
                LD    B, (HL)
                CP    B
                POP   BC
                JR    Z, RAM
                DJNZ  LOOP
;
RIEN:           NOP
                JR    RETOUR
;
RAM:            NOP
                JR    RETOUR
;
MNROM           EQU   0FCC1H
RETOUR:         LD    A, (MNROM)
                LD    HL, 04000H
                CALL  ENASLT
                RET
```

Le programme procède de la manière suivante : il sélectionne tous les slots - primaires et secondaires - un par un. Il écrit une donnée puis la relit. Si la donnée lue est identique à celle écrite, alors le slot actuel contient de la mémoire vive, le programme s'arrête. Dans le cas contraire, on passe au slot suivant.

A vous de compléter les routines « RIEN: » et « RAM: » en fonction de votre application.

L'utilisation :

Sur MSX1, pas de problème, vous avez trouvé de la mémoire vive, utilisez la. Par contre sur MSX2, il faut faire attention à plusieurs choses :

– le disque virtuel peut être implanté dans la mémoire vive que vous venez de trouver. Si tel est le cas, la variable système SLTWRK (0FD09H) contient le numéro du slot dans lequel se trouve le disque virtuel sous la forme FXXXSSPP sachant que :

  F flag 1 = slot secondaire, 0 = slot primaire

  SS numéro de slot secondaire (0-3)

  PP numéro de slot primaire (0-3)

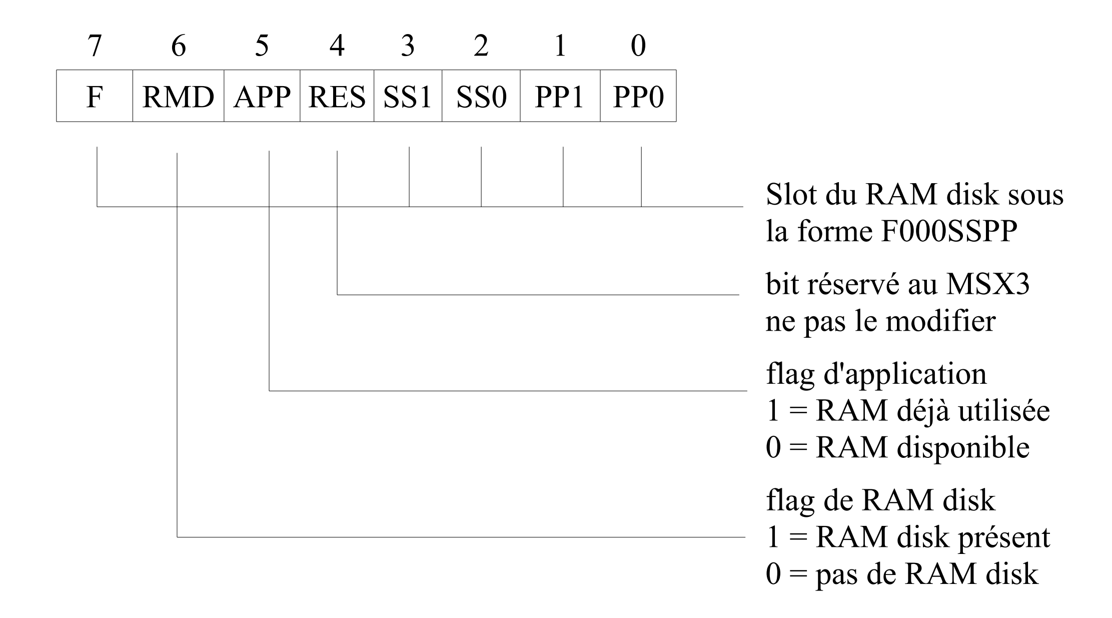
  Nous verrons la signification des bits X plus bas, masquez ces bits pour pouvoir comparer avec le numéro de slot retourné par la routine qui est donnée plus haut. Cependant, il faut faire attention car si SLTWRK donne 0XXX0011 (slot primaire 3), ma routine renverra 1XXX1111 (slot secondaire 3, primaire 3). Le résultat est équivalent lors d'un appel inter-slot ou d'un ENASLT. Pour simplifier la programmation, ma routine donne TOUJOURS le résultat sous forme de slot secondaire même s'il s'agit d'un slot primaire. Il va de soi que s'il n'y a pas de disque virtuel, ces 5 bits seront tous à 0.

Le bit 6 de SLTWRK est un indicateur binaire. S'il est à 0, le disque virtuel n'est pas initialisé; au contraire, le disque virtuel est en place si ce bit est à 1.

Si le disque virtuel est installé, vous pouvez encore disposer de la mémoire vive non utilisée par celui-ci. Il suffit de savoir où s'arrête le disque virtuel. Les cases mémoires 00000H et 00001H contiennent l'adresse de début de la zone mémoire laissée libre par le disque virtuel. Les adresses 00002H à 0007FH ne sont pas utilisées sur MSX2, elles sont réservées aux futures versions du système MSX. Ne pas y toucher. Le disque virtuel commence en 00080H.

Un autre logiciel utilise peut-être déjà la mémoire que vous avez trouvée. Si tel est le cas, le bit 5 de SLTWRK sera à 1. Si le bit est à 0, vous même - si vous décidez d'utiliser la mémoire vive trouvée pour votre application - devez mettre le bit 5 de SLTWRK à 1.

Voici un résumé des fonctions de SLTWRK sur MSX2 :



6.12 DETOURNER LE RESET

Presque tous les MSX vendus en France possèdent un bouton RESET (le V20 de Canon est le seul MSX sans RESET ayant connu une grande diffusion). Le programmeur peut interdire l'accès à son application à l'utilisateur final en détournant le RESET.

Le principe est assez simple. On installe en mémoire vive des codes qui font croire au MSX qu'il ne s'agit pas de RAM mais d'une cartouche. En effet, dans ce dernier cas, le MSX passe toujours la main à la cartouche (sinon le programme en cartouche ne démarrerait pas automatiquement). Etant trompé (comme nous sommes diaboliques), le MSX va donc rendre la main au programme que nous aurons installé auparavant.

On utilise le programme du paragraphe précédent pour trouver la RAM de 04000H à 08000H. Le reste du programme se comprend de lui-même :

```
            ORG   0C000H
;
ENASLT      EQU   00024H
```

```
FIND:       LD    B, 0FH
            LD    HL, 04000H
LOOP:       LD    A, B
            OR    080H
            PUSH  BC
            PUSH  AF
            PUSH  HL
            CALL  ENASLT
            POP   HL
            POP   AF
            LD    (HL), A
            LD    B, (HL)
            CP    B
            POP   BC
            JR    Z, RAM
            DJNZ  LOOP
;
RIEN:       JR    RETOUR
;
RAM:        LD    IX, 04000H
            LD    (IX + 0), 041H
            LD    (IX + 1), 042H
            LD    (IX + 2), 000H
            LD    (IX + 3), 0C1H
            JR    RETOUR
;
MNROM       EQU   0FCC1H
RETOUR:     LD    A, (MNROM)
            LD    HL, 04000H
            CALL  ENASLT
            RET
;
;
;
            ORG   0C100H
;
BREAKX      EQU   000B7H
INITXT      EQU   0006CH
;
            CALL  INITXT
            LD    HL, MESS
            CALL  ECRIRE
WAIT:       CALL  BREAKX
            RET   C
            JR    WAIT
ECRIRE :    LD    A, (HL)
            CP    0
            RET   Z
            CALL  0A2H
            INC   HL
            JR    ECRIRE
MESS:       DB    01BH, 'Y*&'
```

```
                DB      'LE RESET NE REND'
                DB      ' PAS LA MAIN !', 0
```

Dans l'exemple ci-dessus, tapez et assemblez le programme, puis lancez l'exécution en 0C000H. A partir de ce moment, chaque fois que vous appuierez sur le bouton RESET, le programme affichera le message « Le reset ne rend pas la main ! ». Il suffit d'enfoncer les touches CTRL et STOP pour récupérer le contrôle.

Notez que l'on utilise un code ESC pour positionner le curseur (première ligne de « MESS: »).

Une cartouche se distingue des autres supports. En effet, toute cartouche contient « AB » (codes 041H et 042H) comme deux premiers octets (en général c'est aux adresses 04000H et 04001H). Elle renferme ensuite (en 4002H et 4003H) l'adresse de démarrage du programme en cartouche.

6.13 AJOUTER DES MOTS CLEF AU BASIC

Tant qu'on y est, continuons à simuler le fonctionnement d'une cartouche. Vous savez sans doute qu'une cartouche peut parfois contenir des extensions au MSX-Basic. On accède à ces nouvelles instructions par l'instruction CALL (ou « _ ») suivi d'un nom et éventuellement de paramètres.

L'adresse de la routine de traitement des mots clefs est donnée par le 5ème et le 6ème octet (adresse 04004H et 04005H) de la cartouche. A chaque fois que le Basic trouve un CALL, il appelle la routine à cette adresse. Le mot clef se trouve alors sur 16 octets dans la zone des variables système (variable PROCNM, adresses 0FD89H à 0FD98H). Le registre double HL pointe sur le premier caractère non blanc (code 020H) après le mot clef, ce qui permet de récupérer les paramètres. Il faut réactualiser HL de manière à poursuivre l'exécution du programme Basic après traitement du CALL. Il suffit pour cela d'incrémenter HL jusqu'au moment où HL pointe sur un 0 (code de fin de ligne Basic) ou 03AH (code des deux points « : », séparant deux instructions Basic). De plus, votre routine doit toujours rendre la main avec l'indicateur Carry à 0. Dans la routine de traitement, tous les registres peuvent être utilisés (sauf SP, bien sûr). Si le mot clef est inconnu, il faut mettre l'indicateur Carry à 1, puis rendre la main à l'interpréteur par un RET, sans avoir modifié HL.

Dans l'exemple suivant, nous créons un mot clef « FILLSCREEN » qui remplit l'écran (en SCREEN 0 uniquement, 40 ou 80 colonnes) avec le caractère qui suit :

```
10 _FILLSCREEN(« Z ») : BEEP
```

Cet exemple fonctionne parfaitement sur MSX1 et MSX2.

```
                ORG     0C000H
;
ENASLT          EQU     00024H
FIND:           LD      B, 0FH
                LD      HL, 04000H
LOOP:           LD      A, B
                OR      080H
                PUSH    BC
                PUSH    AF
                PUSH    HL
                CALL    ENASLT
```

```
                POP     HL
                POP     AF
                LD      (HL), A
                LD      B, (HL)
                CP      B
                POP     BC
                JR      Z, RAM
                DJNZ    LOOP
;
RIEN:           JR      RETOUR
;
RAM:            LD      IX, 04000H
                LD      (IX + 0), 041H
                LD      (IX + 1), 042H
                LD      (IX + 2), 000H
                LD      (IX + 3), 000H
                LD      (IX + 4), 000H
                LD      (IX + 5), 0C1H
                JR      00000H
;
MNROM           EQU     0FCC1H
RETOUR:         LD      A, (MNROM)
                LD      HL, 04000H
                CALL    ENASLT
                RET
;
;
;
                ORG     0C100H
;
FILVRM          EQU     00056H
                PUSH    HL
                LD      HL, WORD
                LD      DE, 0FD89H
LOOP2:          LD      A, (DE)
                LD      B, (HL)
                CP      B
                JR      NZ, SYNERR
                CP      0
                INC     HL
                INC     DE
                JR      Z, OUT
                JR      LOOP2
;
SYNERR:         POP     HL
                XOR     A
                CCF
                RET
;
OUT:            POP     HL
                PUSH    HL
                LD      A, (HL)
```

```
                CP      028H
                JR      NZ, SYNERR
                INC     HL
                LD      A, (HL)
                CP      022H
                JR      NZ, SYNERR
                INC     HL
                LD      B, (HL)
                INC     HL
                LD      A, (HL)
                CP      022H
                JR      NZ, SYNERR
                INC     HL
                LD      A, (HL)
                CP      029H
                JR      NZ, SYNERR
LOOP3:          INC     HL
                LD      A, (HL)
                CP      020H
                JR      Z, LOOP3
                CP      0
                JR      Z, SUITE
                CP      03AH
                JR      NZ, SYNERR
;
SUITE:          PUSH    HL
                LD      HL, 0
                LD      A, B
                LD      BC, 00800H
                CALL    FILVRM
                POP     HL
                POP     BC
                XOR     A
                RET
;
WORD:           DB      'FILLSCREEN', 000H
```

Après avoir tapé et assemblé le programme ci-dessus, lancez l'exécution en 0C000H. Le MSX fera automatiquement un RESET (pour que la recherche de la cartouche se produise). Lors du retour sur Basic, tapez CLEAR 200, &HC100

A partir de là, vous pouvez à tout moment appeler la nouvelle fonction par un

CALL FILLSCREEN (« caractère »).

Pour créer une autre fonction, il suffit de modifier le programme à partir du label « SUITE »

6.14 MANIPULER LA SOURIS

Je vous propose un sous-programme en langage machine qui permet d'utiliser la souris depuis une application en Basic.

```
                ORG   0C000H
;
EXTROM          EQU   0015FH
NEWPAD          EQU   001ADH
GTTRIG          EQU   000D8H
NWRVRM          EQU   00177H
;
SOURIS:         LD    A, 0CH
                LD    IX, NEWPAD
                CALL  EXTROM
                LD    A, 0DH
                LD    IX, NEWPAD
                CALL  EXTROM
                LD    B, A
                LD    A, (X)
                ADD   A, B
                LD    (X), A
;
                LD    A, 0CH
                LD    IX, NEWPAD
                CALL  EXTROM
                LD    A, 0EH
                LD    IX, NEWPAD
                CALL  EXTROM
                LD    B, A
                LD    A, (Y)
                ADD   A, B
                LD    (Y), A
;
                CALL  SPRITE
                LD    A, 1
                CALL  GTTRIG
                CP    0FFH
                RET   Z
                JR    SOURIS
;
SPRITE:         LD    A, (X)
                LD    HL, 0FA01H
                CALL  NWRVRM
                RET
;
;
X               DB    000H
Y               DB    000H
```

Voici un exemple d'exploitation de la routine ci-dessus en Basic :

```
10 SCREEN 7 : SET PAGE 0, 0 : COLOR 10, 0, 0 : CLS : X = &HC045 : Y = X + 1
20 SPRITE$(0) = CHR$(&HF8) + CHR$(&HC0) + CHR$(&HA0) + CHR$(&H90) + CHR$
(&H88) + CHR$(4) + CHR$(2)
30 PUT SPRITE 0, (100, 100), 7 : DEFUSR=&HC000 : I = USR(0)
110 PSET (PEEK(&HC054)*2, PEEK(&HC055)+1), 14
120 I = USR(0) : GOTO 110
```

ANNEXES

A - LE BIOS

Voici la liste de toutes les routines du Bios classées par ordre alphabétique :

Nom	Adresse	Slot
ATRSCN	00071H	SUB
BASE	00169H	SUB
BASEF	0016DH	SUB
BASRVN	0002BH	ROM
BEEP	000C0H	ROM
« «	0017DH	SUB
BIGFIL	0016BH	ROM
BLTDM	001A9H	SUB
BLTDV	001A1H	SUB
BLTMD	001A5H	SUB
BLTMV	00199H	SUB
BLTVM	00195H	SUB
BLTVD	0019DH	SUB
BLTVV	00191H	SUB
BOXLIN	00081H	SUB
BREAKX	000B7H	ROM
CALATR	00087H	ROM
« «	000FDH	SUB
CALBAS	00159H	ROM
CALLF	00030H	ROM
CALPAT	00084H	ROM

Nom	Adresse	Slot
« «	000F9H	SUB
CALSLT	0001CH	ROM
CGTABL	00004H	ROM
CHGCAP	00132H	ROM
CHGCLR	00062H	ROM
« «	00111H	SUB
CHGET	0009FH	ROM
CHGMDP	001B5H	SUB
CHGMOD	0005FH	ROM
« «	000D1H	SUB
CHGSND	00135H	ROM
CHKNEW	00165H	ROM
CHKRAM	00000H	ROM
CHKSLZ	00162H	ROM
CHPUT	000A2H	ROM
CHRGTR	00010H	ROM
CHSNS	0009CH	ROM
CKCNTC	000BDH	ROM
CLRSPR	00069H	ROM
« «	000F5H	SUB
CLRTXT	00119H	SUB

CLS	000C3H	ROM
« «	00115H	SUB
CNVCHR	000ABH	ROM
COLOR	00155H	SUB
DCOMPR	00020H	ROM
DELLNO	00121H	SUB
DISSCR	00041H	ROM
DOBOXF	00079H	SUB
DOGRPH	00085H	SUB
DOLINE	0007DH	SUB
DOWNC	00108H	ROM
« «	000B5H	SUB
DSPFNK	000CFH	ROM
« «	0011DH	SUB
ENASCR	00044H	ROM
ENASLT	00024H	ROM
EOL	00168H	ROM
ERAFNK	000CCH	ROM
EXTROM	0015FH	ROM
FETCHC	00114H	ROM
FILVRM	00056H	ROM
FNKSB	000C9H	ROM
FORMAT	00147H	ROM
GETPAT	00105H	SUB
GETPLT	00149H	SUB
GETPUT	001B1H	SUB
GETVCP	00150H	ROM
GETVC2	00153H	ROM
GETYPR	00028H	ROM
GICINI	00090H	ROM
GLINE	00075H	SUB
GRPPRT	0008DH	ROM
« «	00089H	SUB
GSPSIZ	0008AH	ROM
« «	00101H	SUB
GTASPC	00126H	ROM
GTPAD	000DBH	ROM
GTPDL	000DEH	ROM
GTSTCK	000D5H	ROM
GTTRIG	000D8H	ROM
INIFNK	0003EH	ROM
INIGRP	00072H	ROM
« «	000DDH	SUB
INIMLT	00075H	ROM
« «	000E1H	SUB
INIPLT	00141H	SUB
INITIO	0003BH	ROM
INITXT	0006CH	ROM
« «	000D5H	SUB
INIT32	0006FH	ROM
INIT32	000D9H	SUB
INLIN	000B1H	ROM
INSLN0	00125H	SUB
ISCNTC	000BAH	ROM
ISFLIO	0014AH	ROM
KEYINT	00038H	ROM
KILBUF	00156H	ROM
KNJPRT	001BDH	SUB
KYKLOK	00135H	SUB
LDIRMV	00059H	ROM
LDIRVM	0005CH	ROM
LEFTC	000FFH	ROM
« «	000ADH	SUB
LFTQ	000F6H	ROM
LPTOUT	000A5H	ROM
LPTSTT	000A8H	ROM
MAPXYC	00111H	ROM
« «	00091H	SUB
MSXVER	0002DH	ROM
NEWPAD	001ADH	SUB
NMI	00066H	ROM
NRDVRM	00174H	ROM
NSETCX	00123H	ROM
NSETRD	0016EH	ROM

NSTWRT	00171H	ROM
NVBXFL	000CDH	SUB
NVBXLN	000C9H	SUB
NWRVRM	00177H	ROM
OUTLDP	0014DH	ROM
OUTDO	00018H	ROM
PAINT	00069H	SUB
PNTINI	00129H	ROM
PINLIN	000AEH	ROM
POSIT	000C6H	ROM
PSET	0006DH	SUB
PUTCHR	00139H	SUB
PUTQ	000F9H	ROM
PUTSPR	00151H	SUB
PUTVRM	00129H	SUB
PYSDIO	00144H	ROM
QINLIN	000B4H	ROM
RDPSG	00096H	ROM
RDSLT	0000CH	ROM
RDVDP	0013EH	ROM
RDVRM	0004AH	ROM
« «	0010DH	SUB
READC	0011DH	ROM
« «	00095H	SUB
REDCLK	001F5H	SUB
RIGHTC	000FCH	ROM
« «	000A5H	SUB
PROMPT	00181H	SUB
RSLREG	00138H	ROM
RSTPLT	00145H	SUB
SCALXY	0010EH	ROM
« «	0008DH	SUB
SCANL	0012FH	ROM
« «	000C5H	SUB
SCANR	0012CH	ROM
« «	000C1H	SUB
SCOPY	0018DH	SUB
SCREEN	00159H	SUB
SDFSCR	00185H	SUB
SETATR	0011AH	ROM
« «	00099H	SUB
SETC	00120H	ROM
« «	0009DH	SUB
SETGRP	0007EH	ROM
« «	000EDH	SUB
SETMLT	00081H	ROM
« «	000F1H	SUB
SETPAG	0013DH	SUB
SETPLT	0014DH	SUB
SETRD	00050H	ROM
SETS	00179H	SUB
SETSCR	00189H	SUB
SETTXT	00078H	ROM
« «	000E5H	SUB
SETT32	0007BH	ROM
« «	000E9H	SUB
SETWRT	00053H	ROM
SNSMAT	00141H	ROM
STMOTR	000F3H	ROM
STOREC	00117H	ROM
STRTMS	00099H	ROM
SUBROM	0015CH	ROM
SYNCHR	00008H	ROM
TAPIN	000E4H	ROM
TAPIOF	000E7H	ROM
TAPION	000E1H	ROM
TAPOOF	000F0H	ROM
TAPOON	000EAH	ROM
TAPOUT	000EDH	ROM
TDOWNC	0010BH	ROM
« «	000B1H	SUB
TLEFTC	000A9H	SUB
TOTEXT	000D2H	ROM
TRIGHT	000A1H	SUB

TUPC	00105H	ROM	VPOKE	00171H	SUB
« «	000B9H	SUB	WIDTHS	0015DH	SUB
UPC	00102H	ROM	WRTCLK	001F9H	SUB
« «	000BDH	SUB	WRTPSG	00093H	ROM
VDP	00161H	SUB	WRTSLT	00014H	ROM
VDPF	00165H	SUB	WRTVDP	00047H	ROM
VDPSTA	00131H	SUB	« «	0012DH	SUB
VDP.DR	00006H	ROM	WRTVRM	0004DH	ROM
VDP.DW	00007H	ROM	« «	00109H	ROM
VPEEK	00175H	SUB	WSLREG	0013BH	ROM

B - LES VARIABLES SYSTEME

Liste des variables système dans l'ordre alphabétique :

Nom	Adresse	Longueur	Type
ACPAGE	0FAF6H	1	MSX2
ARG	0F847H	16	MSX1
ARYTAB	0F6C4H	2	MSX1
ARYTA2	0F7B5H	2	MSX1
ASPCT1	0F40BH	2	MSX1
ASPCT2	0F40DH	2	MSX1
ASPECT	0F931H	2	MSX1
ATRBAS	0F928H	2	MSX1
ATRBYT	0F3F2H	1	MSX1
AUTFLG	0F6AAH	1	MSX1
AUTINC	0F6ADH	2	MSX1
AUTLIN	0F6ABH	2	MSX1
AVCSAV	0FAF7H	1	MSX2
BAKCLR	0F3EAH	1	MSX1
BASROM	0FBB1H	1	MSX1
BDRCLR	0F3EBH	1	MSX1
BOTTOM	0FC48H	2	MSX1
BRDATR	0FCB2H	1	MSX1
BUF	0F55EH	256	MSX1
BUFEND	0FC18H	0	MSX1
BUFMIN	0F55DH	1	MSX1

Nom	Adresse	Longueur	Type
CAPST	0FCABH	1	MSX1
CASPRV	0FCB1H	1	MSX1
CENCNT	0F933H	1	MSX1
CGPBAS	0F924H	2	MSX1
CGPNT	0F91FH	2	MSX1
CHRCNT	0FAF9H	3	MSX2
CLIKFL	0FBD9H	1	MSX1
CLIKSW	0F3DBH	1	MSX1
CLINEF	0F935H	1	MSX1
CLMSLT	0F3B2H	1	MSX1
CLOC	0F92AH	2	MSX1
CMASK	0F92CH	1	MSX1
CNPNTS	0F936H	2	MSX1
CNSDFG	0F3DEH	1	MSX1
CODSAV	0FBCCH	1	MSX1
CONLO	0F66AH	8	MSX1
CONSAV	0F668H	1	MSX1
CONTXT	0F666H	2	MSX1
CONTYP	0F669H	1	MSX1
CPCNT	0F939H	2	MSX1
CPCNT8	0F93BH	2	MSX1

CPLOTF	0F938H	1	MSX1	ERRFLG	0F414H	1	MSX1
CRCSUM	0H93DH	2	MSX1	ERRLIN	0F6B3H	2	MSX1
CRTCNT	0F3B1H	1	MSX1	ERRTXT	0F6B7H	2	MSX1
CS120	0F3FCH	10	MSX1	ESCCNT	0FCA7H	1	MSX1
CSAVEA	0F942H	2	MSX1	EXBRSA	0FAF8H	1	MSX2
CSAVEM	0F944H	1	MSX1	EXPTBL	0FCC1H	4	MSX1
CSCLXY	0F941H	1	MSX1	FACLO	0F7F8H	interne	MSX1
CSRSW	0FCA9H	1	MSX1	FBUFFER	0F7C5H	43	MSX1
CSRX	0F3DDH	1	MSX1	FILNAM	0F866H	11	MSX1
XSRY	0F3DCH	1	MSX1	FILNM2	0F871H	11	MSX1
CSTCNT	0F93FH	2	MSX1	FILTAB	0F860H	2	MSX1
CSTYLE	0FCAAH	1	MSX1	FLBMEM	0FCAEH	1	MSX1
CURLIN	0F41CH	2	MSX1	FLGINP	0F6A6H	1	MSX1
CXOFF	0F945H	2	MSX1	FKNFLG	0FBCEH	10	MSX1
CYOFF	0F947H	2	MSX1	FNKSTR	0F87FH	160	MSX1
DAC	0F7F6H	16	MSX1	FNKSWI	0FBCDH	1	MSX1
DATLIN	0F6A3H	2	MSX1	FORCLR	0F3E9H	1	MSX1
DATPTR	0F6C8H	2	MSX1	FRCNEW	0F3F5H	1	MSX1
DECCNT	0F7F4H	1	MSX1	FRETOP	0F69BH	2	MSX1
DECTMP	0F7F0H	2	MSX1	FSTPOS	0FBCAH	2	MSX1
DECTM2	0F7F2H	2	MSX1	FUNACT	0F7BAH	2	MSX1
DEFTBL	0F6CAH	26	MSX1	GETPNT	0F3FAH	2	MSX1
DEVICE	0FD99H	1	MSX1	GRPACX	0FCB7H	2	MSX1
DIMFLG	0F662H	1	MSX1	GRPACY	0FCB9H	2	MSX1
DORES	0F664H	1	MSX1	GRPATR	0F3CDH	2	MSX1
DONUM	0F665H	1	MSX1	GRPCGP	0F3CBH	2	MSX1
DOT	0F6B5H	2	MSX1	GRPCOL	0F3C9H	2	MSX1
DPPAGE	0FAF5H	1	MSX2	GRPHED	0FCA6H	1	MSX1
DRWANG	0FCBDH	1	MSX1	GRPNAM	0F3C7H	2	MSX1
DRWFLG	0FCBBH	1	MSX1	GRPPAT	0F3CFH	2	MSX1
DRWSCL	0FCBCH	1	MSX1	GXPOS	0FCB3H	2	MSX1
DSCPTR	0F699H	interne	MSX1	GYPOS	0FCB5H	2	MSX1
DSCTMP	0F698H	3	MSX1	HEADER	0F40AH	1	MSX1
ENDBUF	0F660H	1	MSX1	HIGH	0F408H	2	MSX1
ENDFOR	0F6A1H	2	MSX1	HIMEM	0FC4AH	2	MSX1
ENDPRG	0F40FH	5	MSX1	HOLD	0F83EH	8	MSX1
ENSTOP	0FBB0H	1	MSX1	HOLD2	0F836H	8	MSX1

HOLD8	0F806H	48	MSX1	MLTNAM	0F3D1H	2	MSX1
INSFLG	0FCA8H	1	MSX1	MLTPAT	0F3D9H	2	MSX1
INTCNT	0FCA2H	2	MSX1	MNROM	0FCC1H	1	MSX2
INTFLG	0FC9BH	1	MSX1	MODE	0FAFCH	1	MSX2
INTVAL	0FCA0H	2	MSX1	MOVCNT	0F951H	2	MSX1
JIFFY	0FC9EH	2	MSX1	MUSICF	0FB3FH	1	MSX1
KANAMD	0FCADH	1	MSX1	NAMBAS	0F922H	2	MSX1
KANAST	0FCACH	1	MSX1	NEWKEY	0FBE5H	11	MSX1
KBUF	0F41FH	316	MSX1	NLONLY	0F87CH	1	MSX1
KEYBUF	0FBF0H	40	MSX1	NOFUNS	0F7B7H	1	MSX1
LFPROG	0F954H	1	MSX1	NTMSXP	0F417H	1	MSX1
LINL32	0F3AFH	1	MSX1	NULBUF	0F862H	2	MSX1
LINL40	0F3AEH	1	MSX1	OLDKEY	0FBDAH	11	MSX1
LINLEN	0F3B0H	1	MSX1	OLDLIN	0F6BEH	2	MSX1
LINTTB	0FBB2H	24	MSX1	OLDSCR	0FCB0H	1	MSX1
LINWRK	0FC18H	40	MSX1	OLDTXT	0F6C0H	2	MSX1
LOGOPR	0FB02H	1	MSX2	ONEFLG	0F6BBH	1	MSX1
LOHADR	0F94BH	2	MSX1	ONELIN	0F6B9H	2	MSX1
LOHCNT	0F94DH	2	MSX1	ONGSBF	0FBD8H	1	MSX1
LOHDIR	0F94AH	1	MSX1	OPRTYP	0F664H	0	MSX1
LOHMSK	0F949H	1	MSX1	PADX	0FC9DH	1	MSX1
LOW	0F406H	2	MSX1	PADY	0FC9CH	1	MSX1
LOWLIM	0FCA4H	1	MSX1	PARM1	0F6E8H	100	MSX1
LPTPOS	0F415H	1	MSX1	PARM2	0F750H	100	MSX1
MAXDEL	0F92FH	2	MSX1	PATBAS	0F926H	2	MSX1
MAXFIL	0F85FH	1	MSX1	PATWRK	0FC40H	8	MSX1
MAXUPD	0F3ECH	3	MSX1	PDIREC	0F953H	1	MSX1
MCLFLG	0F958H	1	MSX1	PLYCNT	0FB40H	1	MSX1
MCLLEN	0FB3BH	1	MSX1	PRMFLG	0F7B4H	1	MSX1
MCLPTR	0FB3CH	2	MSX1	PRMLEN	0F6E6H	2	MSX1
MCLTAB	0F956H	2	MSX1	PRMLN2	0F74EH	2	MSX1
MEMSIZ	0F672H	2	MSX1	PRMPRV	0F74CH	2	MSX1
MINDEL	0F92DH	2	MSX1	PRMSTK	0F6E4H	2	MSX1
MINUPD	0F3EFH	3	MSX1	PROCNM	0FD89H	16	MSX1
MLTATR	0F3D7H	2	MSX1	PRSCNT	0FB35H	1	MSX1
MLTCGP	0F3D5H	2	MSX1	PTRFIL	0F864H	2	MSX1
MLTCOL	0F3D3H	2	MSX1	PTRFLG	0F6A9H	1	MSX1

PUTPNT	0F3F8H	2	MSX1
QUEBAK	0F971H	4	MSX1
QUETAB	0F959H	24	MSX1
QUEUEN	0FB3EH	1	MSX1
QUEUES	0F3F3H	2	MSX1
RAWPRT	0F418H	1	MSX1
REPCNT	0F3F7H	1	MSX1
REQSTP	0FC6AH	1	MSX1
RG0SAV	0F3DFH	1	MSX1
RG1SAV	0F3E0H	1	MSX1
RG2SAV	0F3E1H	1	MSX1
RG3SAV	0F3E2H	1	MSX1
RG4SAV	0F3E3H	1	MSX1
RG5SAV	0F3E4H	1	MSX1
RG6SAV	0F3E5H	1	MSX1
RG7SAV	0F3E6H	1	MSX1
ROMA	0FAFAH	2	MSX2
RNDX	0F857H	8	MSX1
RSFCB	0FB04H	1	MSX2
RSIQLN	0FB06H	1	MSX2
RS2IQ	0FAF5H	64	MSX1
RTPROG	0F955H	1	MSX1
RTYCNT	0FC9AH	1	MSX1
RUNBNF	0FCBEH	1	MSX1
RUNFLG	0F866H	0	MSX1
SAVEND	0F87DH	2	MSX1
SAVENT	0FCBFH	2	MSX1
SAVSP	0FB36H	2	MSX1
SAVSTK	0F6B1H	2	MSX1
SAVTXT	0F6AFH	2	MSX1
SAVVOL	0FB39H	2	MSX1
SCNCNT	0F3F6H	1	MSX1
SCRMOD	0FCAFH	1	MSX1
SFTKEY	0FBEBH	interne	MSX1
SKPCNT	0F94FH	2	MSX1
SLTATR	0FCC9H	64	MSX1
SLTTBL	0FCC5H	4	MSX1
SLTWRK	0FD09H	128	MSX1
STATFL	0F3E7H	1	MSX1
STKTOP	0F674H	2	MSX1
STREND	0F6C6H	2	MSX1
SUBFLG	0F6A5H	1	MSX1
SUBROM	0FAF8H	1	MSX2
SWPTMP	0F7BCH	8	MSX1
TEMP	0F6A7H	2	MSX1
TEMP2	0F6BCH	2	MSX1
TEMP3	0F69DH	2	MSX1
TEMP8	0F69FH	2	MSX1
TEMP9	0F7B9H	2	MSX1
TEMPPT	0F678H	2	MSX1
TEMPST	0F67AH	30	MSX1
TOCNT	0FB03H	1	MSX2
TRCFLG	0F7C4H	1	MSX1
TRGFLG	0F3E8H	1	MSX1
TRPTBL	0FC4CH	78	MSX1
TTYPOS	0F661H	1	MSX1
TXTATR	0F3B9H	2	MSX1
TXTCGP	0F3B7H	2	MSX1
TXTCOL	0F3B5H	2	MSX1
TXTNAM	0F3B3H	2	MSX1
TXTPAT	0F3BBH	2	MSX1
TXTTAB	0F676H	2	MSX1
T32ATR	0F3C3H	2	MSX1
T32CGP	0F3C1H	2	MSX1
T32COL	0F3BFH	2	MSX1
T32NAM	0F3BDH	2	MSX1
T32PAT	0F3C5H	2	MSX1
USRTAB	0F39AH	20	MSX1
VALTYP	0F663H	1	MSX1
VARTAB	0F6C2H	2	MSX1
VCBA	0FB41H	37	MSX1
VCBB	0FB66H	37	MSX1
VCBC	0FB8BH	37	MSX1
VLZADR	0F419H	2	MSX1

VLZDAT	0F41BH	1	MSX1	VOICEN	0FB38H	1	MSX1
VOICAQ	0F975H	128	MSX1	WINWID	0FCA5H	1	MSX1
VOICBQ	0F9F5H	128	MSX1	XSAVE	0FAFEH	2	MSX2
VOICBC	0FA75H	128	MSX1	YSAVE	0FB00H	2	MSX2

0 : Adresse partagée par plusieurs variables.

C - LES HOOKS

Voici la liste des hooks classés dans l'ordre alphabétique :

Nom	Adresse	Nom	Adresse	Nom	Adresse	Nom	Adresse
H.ATTR	0FE1CH	H.DSKI	0FE17H	H.GETP	0FE4EH	H.MKS$	0FE35H
H.BAKU	0FEADH	H.DSKO	0FDEFH	H.GONE	0FF43H	H.NAME	0FDF9H
H.BINL	0FE76H	H.DSPC	0FDA9H	H.INDS	0FEA8H	H.NEWS	0FF3EH
H.BINS	0FE71H	H.DSPF	0FDB3H	H.INIP	0FDC7H	H.NMI	0FDD6H
H.BUFL	0FF8EH	H.EOF	0FEA3H	H.INLI	0FDE5H	H.NODE	0FEB7H
H.CHGE	0FDC2H	H.ERAC	0FDAEH	H.IPL	0FE03H	H.NOFO	0FE58H
H.CHPU	0FDA4H	H.ERAF	0FDB8H	H.ISFL	0FEDFH	H.NOTR	0FF34H
H.CHRG	0FF48H	H.ERRF	0FF02H	H.ISMI	0FF7FH	H.NTFL	0FE62H
H.CLEA	0FED0H	H.ERRO	0FFB1H	H.ISRE	0FF2AH	H.NTFN	0FF2FH
H.CMD	0FE0DH	H.ERRP	0FEFDH	H.KEYC	0FDCCH	H.NTPL	0FF6BH
H.COMP	0FF57H	H.EVAL	0FF70H	H.KEYI	0FD9AH	H.NULO	0FE5DH
H.COPY	0FE08H	H.FIEL	0FE2BH	H.KILL	0FDFEH	H.OKNO	0FF75H
H.CRDO	0FEE9H	H.FILE	0FE7BH	H.KYEA	0FDD1H	H.ONGO	0FDEAH
H.CRUN	0FF20H	H.FILO	0FE85H	H.LIST	0FF89H	H.OUTD	0FEE4H
H.CRUS	0FF25H	H.FINE	0FF1BH	H.LOC	0FE99H	H.PARD	0FEB2H
H.CVD	0FE49H	H.FING	0FF7AH	H.LOF	0FE9EH	H.PHYD	0FFA7H
H.CVI	0FE3FH	H.FINI	0FF16H	H.LOPD	0FED5H	H.PINL	0FDDBH
H.CVS	0FE44H	H.FINP	0FF5CH	H.LPTO	0FFB6H	H.PLAY	0FFC5H
H.DEVN	0FEC1H	H.FORM	0FFACH	H.LPTS	0FFBBH	H.POSD	0FEBCH
H.DGET	0FFE8H	H.FPOS	0FEA8H	H.LSET	0FE21H	H.PRGE	0FEF8H
H.DIRD	0FF11H	H.FRET	0FF9DH	H.MAIN	0FF0CH	H.PRTF	0FF52H
H.DOGR	0FEF3H	H.FRME	0FF66H	H.MERG	0FE67H	H.PTRG	0FFA2H
H.DSKC	0FEEEH	H.FRQI	0FF93H	H.MKD$	0FE3AH	H.QINL	0FDE0H
H.DSKF	0FE12H	H.GEND	0FEC6H	H.MKI$	0FE30H	H.READ	0FF07H

H.RETU	0FF4DH	H.SAVD	0FE94H	H.SETF	0FE53H	H.TIMI	0FD9FH
H.RSET	0FE26H	H.SAVE	0FE6CH	H.SETS	0FDF4H	H.TOTE	0FDBDH
H.RSLF	0FE8FH	H.SCNE	0FF98H	H.SNGF	0FF39H	H.TRMN	0FF61H
H.RUNC	0FECBH	H.SCRE	0FFC0H	H.STKE	0FEDAH	H.WIDT	0FF84H

D - LES REGISTRES DU V9938

Contenu des registres du VDP 9938 du MSX2 :

registre 0	0	DG	IE2	IE1	M5	M4	M3	EV
registre 1	0	BL	IE0	M1	M2	0	SI	MAG
registre 2	0	N16	N15	N14	N13	N12	N11	N10
registre 3	C13	C12	C11	C10	C9	C8	C7	C6
registre 4	0	0	F16	F15	F14	F13	F12	F11
registre 5	S14	S13	S12	S11	S10	S9	S8	S7
registre 6	0	0	P16	P15	P14	P13	P12	P11
registre 7	TC3	TC2	TC1	TC0	BD3	BD2	BD1	BD0
registre 8	MS	LP	TP	CB	VR	0	SPD	BW
registre 9	LN	0	S1	S0	IL	E0	NT	DC
registre 10	0	0	0	0	0	C16	C15	C14
registre 11	0	0	0	0	0	0	S16	S15
registre 12	T23	T22	T21	T20	BC3	BC2	BC1	BC0
registre 13	ON3	ON2	ON1	ON0	OF3	OF2	OF1	OF0
registre 14	0	0	0	0	0	V16	V15	V14
registre 15	0	0	0	0	ST3	ST2	ST1	ST0
registre 16	0	0	0	0	CC3	CC2	CC1	CC0
registre 17	AII	0	RS5	RS4	RS3	RS2	RS1	RS0
registre 18	V3	V2	V1	V0	H3	H2	H1	H0
registre 19	IL7	IL6	IL5	IL4	IL3	IL2	IL1	IL0
registre 20	0	0	0	0	0	0	0	0
registre 21	0	0	1	1	1	0	1	1
registre 22	0	0	0	0	0	1	0	1
registre 23	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0
registre 32	SX7	SX6	SX5	SX4	SX3	SX2	SX1	SX0
registre 33	0	0	0	0	0	0	0	SX8

registre 34	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0
registre 35	0	0	0	0	0	0	SY9	SY8
registre 36	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0
registre 37	0	0	0	0	0	0	0	DX8
registre 38	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0
registre 39	0	0	0	0	0	0	DY9	DY8
registre 40	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0
registre 41	0	0	0	0	0	0	0	NX8
registre 42	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0
registre 43	0	0	0	0	0	0	NY9	NY8
registre 44	CL7	CL6	CL5	CL4	CL3	CL2	CL1	CL0
registre 45	0	MXC	MXD	MXS	DIY	DIX	EQ	MAJ
registre 46	CM3	CM2	CM1	CM0	LO3	LO2	LO1	LO0

Liste alphabétique :

nom	registre	fonction
AII	17	1 = auto incrémentation / 0 = normal
BC3 - BC0	12	en cas de clignotement, couleur 2nde partie
BD3- BD0	7	« back drop color »
BL	1	1 = affichage autorisé / 0 = affichage interdit
BW	8	1 = noir et blanc (32 teintes) / 0 = couleur
CB	8	1 = bus de couleur en entrée / 0 = en sortie
CC3 - CC0	16	n° de couleur lors d'un accès palette
CL3 - CL0	44	couleur lors d'une commande
CM3 - CM0	46	commande à exécuter
C13 - C6	3	adresse table des couleurs (11 bits de ...
C16 - C14	10	... poids fort sur les 17 de l'adresse)
DC	9	1 = DTCLK en entrée / 0 = DTCLK en sortie
DG	0	1 = bus de couleur en entrée / 0 = en sortie
DIX	45	direction horizontale 1 = gauche / 0 = droite
DIY	45	direction verticale 1 = haut / 0= bas
DO7 - DO0	23	décalage vertical de l'écran (0-255)
DX7 - DX0	36	abscisse destination (8 bits poids faible)
DX8	37	abscisse destination (bit poids fort)
DY7 - DY0	38	ordonnée destination (8 bits poids faible)
DY9 et DY8	39	ordonnée destination (2 bits poids fort)
EQ	45	SRCH : 1 = retour sur couleur de bordure, 0 = retour sur couleur autre que

		bordure
E0	9	1 = affichage alterné de 2 pages / 0 = normal
F16 - F11	4	adresse table des formes (6 bits poids forts sur 17)
H3 - H0	18	set adjust horizontal (7 = gauche, 0 = centre, 8 = droite)
IE0	1	1 = interruption scan horizontal OK / 0 = interdit
IE1	0	1 = interruption scan horizontal OK / 0 = interdit
IE2	0	1 = interruption stylo optique OK / 0 = interdit
IL	9	1 = affichage entrelacé / 0 = non entrelacé
IL7 - IL0	19	n° de ligne où l'interruption doit se déclencher lors d'un scan
LN	9	1 = 212 points / 0 = 192 points
LO3 - LO0	46	opérateur logique lors d'une commande
LP	8	1 = stylo optique autorisé / 0 = stylo interdit
MAG	1	1 = sprites agrandis / 0 = sprites normaux
MAJ	45	côté le plus long 1 = vertical / 0 = horizontal
MS	8	1 = souris autorisée / 0 = souris interdite
MXC	45	inutilisé sur MSX2
MXD	45	destination : 1 = VRAM étendue/ 0 = VRAM
MXS	45	source : 1 = VRAM étendue/ 0 = VRAM
M2 - M1	1	2 bits de poids faible du mode d'écran
M5 - M3	0	3 bits de poids fort du mode d'écran
NT	9	1 = PAL (313 lignes) / 0 = NTSC (262 lignes)
NX7 - NX0	40	longueur (8 bits poids faible)
NX8	41	longueur (bit poids fort)
NY7 - NY0	42	hauteur (8 bits poids faible)
NY9 et NY8	43	hauteur (2 bits poids fort)
N16 - N10	2	adresse table des noms (7 bits de poids fort sur 17)
OF3 - OF0	13	en cas de clignotement, temps éteint
ON3 - ON0	13	en cas de clignotement, temps allumé
P16 - P11	6	adresse table génératrice des sprites (6 bits de poids fort sur 17)
RS5 - RS0	17	n° de registre lors d'un adressage indirect
SI	1	1 = sprites 16*16 / 0 = sprites 8*8
SPD	8	1 = sprites interdits / 0 = sprites autorisés
ST3 - ST0	15	n° registre statut lors d'une lecture
SX7 - SX0	32	abscisse source (8 bits poids faible)
SX8	33	abscisse source (bit poids fort)
SY7 - SY0	34	ordonnée source (8 bits poids faible)
SY9 et SY8	35	ordonnée source (2 bits poids fort)

S1 et S0	9	choix du mode simultané ou pas
S14 - S7	6	adresse table des attributs des sprites (10 bits...
S16 et S15	11	... de poids fort sur 17)
TC3 - TC0	7	couleur du texte en SCREEN 0 et 1
TP	8	couleur 0 égale à la couleur de la palette
T23 - T20	12	en cas de clignotement, couleur 1ère partie
V3 - V0	18	set adjust vertical (8 = bas, 0 = milieu, 7 = haut)
V16 - V14	14	3 bits de poids fort de l'adresse VRAM
VR	8	type de VRAM 1 = 64*1 bit ou 64*4 bits / 0 = 16*1bit ou 16*4bits

E - CARTES MEMOIRES VIDEO

La carte mémoire vidéo par défaut pour chaque mode graphique :

<u>SCREEN 0</u> 40 colonnes MSX1/MSX2

Table des noms :	00000H - 003BFH	960 octets
Table des formes :	00800H - 00FFFH	2048 octets
Table de palette (MSX2) :	00400H - 0041FH	32 octets

<u>SCREEN 0</u> 80 colonnes MSX2

Table des noms 24 lignes :	00000H - 0077FH	1920 octets
26,5 lignes :	00000H - 0086FH	2160 octets
Table des couleurs 24 lignes :	00800H - 008EFH	240 octets
26,5 lignes :	00800H - 0090DH	270 octets
Table des formes :	01000H - 017FFH	2048 octets
Table de palette (MSX2) :	00F00H - 00F1FH	32 octets

<u>SCREEN 1</u> 32 colonnes MSX1/MSX2

Table des noms :	01800H - 01AFFH	768 octets
Table des couleurs :	02000H - 0201FH	32 octets
Table des formes :	00000H - 007FFH	2048 octets
Table des attributs sprite :	01B00H - 01B7FH	128 octets
Table des formes de sprite :	03800H - 03FFFH	2048 octets

Table de palette (MSX2) :	02020H - 0203FH	32 octets

<u>SCREEN 2</u> MSX1/MSX2

Table des noms :	01800H - 01AFFH	768 octets
Table des couleurs :	02000H - 037FFH	6144 octets
Table des formes :	00000H - 017FFH	6144 octets
Table des attributs sprite :	01B00H - 01B7FH	128 octets
Table des formes de sprite :	03800H - 03FFFH	2048 octets
Table de palette (MSX2) :	02020H - 0203FH	32 octets

<u>SCREEN 3</u> MSX1/MSX2

Table des noms :	00800H - 00AFFH	768 octets
Table des formes :	00000H - 007FFH	2048 octets
Table des attributs sprite :	01B00H - 01B7FH	128 octets
Table des formes de sprite :	03800H - 03FFFH	2048 octets
Table de palette (MSX2) :	02020H - 0203FH	32 octets

<u>SCREEN 4</u> MSX2

Table des noms :	01800H - 01AFFH	768 octets
Table des couleurs :	02000H - 037FFH	6144 octets
Table des formes :	00000H - 017FFH	6144 octets
Table des attributs sprite :	01E00H - 01E7FH	128 octets
Table des formes de sprite :	03800H - 03FFFH	2048 octets
Table des couleurs sprite :	01C00H - 0D7FFH	512 octets
Table de palette :	01E80H - 01E9FH	32 octets

<u>SCREEN 5</u> MSX2

Table des noms 192 lignes :	00000H - 05FFFH	24576 octets
212 lignes :	00000H - 069FFH	27136 octets
Table des attributs sprite :	07600H - 0767FH	128 octets
Table des formes de sprite :	07800H - 07FFFH	2048 octets
Table des couleurs sprite :	07400H - 075FFH	512 octets
Table de palette :	07680H - 0769FH	32 octets

<u>SCREEN 6</u> MSX2

Table des noms 192 lignes :	00000H - 05FFFH	24576 octets
212 lignes :	00000H - 069FFH	27136 octets
Table des attributs sprite :	07600H - 0767FH	128 octets
Table des formes de sprite :	07800H - 07FFFH	2048 octets
Table des couleurs sprite :	07400H - 075FFH	512 octets
Table de palette :	07680H - 0769FH	32 octets

<u>SCREEN 7</u> MSX2

Table des noms 192 lignes :	00000H - 0BFFFH	49152 octets
212 lignes :	00000H - 0D3FFH	54272 octets
Table des attributs sprite :	0FA00H - 0FA7FH	128 octets
Table des formes de sprite :	0F000H - 0F7FFH	2048 octets
Table des couleurs sprite :	0F800H - 0F9FFH	512 octets
Table de palette :	0FA80H - 0FA9FH	32 octets

<u>SCREEN 8</u> MSX2

Table des noms 192 lignes :	00000H - 0BFFFH	49152 octets
212 lignes :	00000H - 0D3FFH	54272 octets
Table des attributs sprite :	0FA00H - 0FA7FH	128 octets
Table des formes de sprite :	0F000H - 0F7FFH	2048 octets
Table des couleurs sprite :	0F800H - 0F9FFH	512 octets
Table de palette :	0FA80H - 0FA9FH	32 octets

F - TABLE ASCII ETENDUE

Vous trouverez dans le tableau récapitulatif suivant le jeu de caractères MSX. Il suit un code ASCII étendu :

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	•	PRW	BRK	•	EFF	NTW	BIP	BS	TAB	CDW	HOM	CLS	RET	EOL	•
1	•	•	INS	•	•	CLL	•	•	SEL	•	•	ESC	CUP	CRT	CLT	CDW
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	¦	}	~	DEL
8	Ç	ü	é	â	ä	à	å	ç	ê	ë	è	ï	î	ì	Ä	Å
9	É	æ	Æ	ô	ö	ò	û	ù	ÿ	Ö	Ü	¢	£	¥	₧	ƒ
A	á	í	ó	ú	ñ	Ñ	ª	º	¿	⌐	¬	½	¼	¡	«	»
B	Ã	ã	Ĩ	ĩ	Õ	õ	Ũ	ũ	Ĳ	ĳ	¾	∽	◇	‰	¶	§
C	▂	▚	■	▔	▪	■	▎	▞	▮	▌	█	⁄⁄	⧵⧵	▼	▲	▶
D	◀	⧗	⧓	▘	▗	▝	▖	※	Δ	‡	ω	█	▬	▌	▐	▀
E	α	β	Γ	π	Σ	σ	µ	τ	Φ	θ	Ω	δ	∞	ø	∈	∩
F	≡	±	≥	≤	⌠	⌡	÷	≈	°	∙	·	√	ⁿ	²	■	LVC

PRW : place le curseur sur le mot précédente
EFF : efface la ligne à droite du curseur
NTW : place le curseur sur le mot suivant
CDW : descend le curseur d'une ligne
HOM : HOME, positionnement du curseur en haut à gauche
EOL : place le curseur en fin de ligne
INS : INS, passage en mode insertion et retour
CLL : efface la ligne ou se trouve le curseur
SEL : SELECT, comme la touche du même nom
CUP : curseur vers le haut
CRT : curseur vers la droite
CLT : curseur vers la gauche
CDW : curseur vers le bas
LVC : curseur vivant, image du curseur

Pratique du MSX2

Pratique du MSX2

par Eric VON ASCHEBERG

retapé par Granced
(original aimablement fourni par Metalion)

pour la communauté MSX française